

Open Call-by-Value (Extended Version)

Beniamino Accattoli¹ and Giulio Guerrieri²

¹ INRIA, UMR 7161, LIX, École Polytechnique, beniamino.accattoli@inria.fr,

² Aix Marseille Université, CNRS, Centrale Marseille, I2M UMR 7373,
F-13453 Marseille, France, giulio.guerrieri@univ-amu.fr

Abstract. The elegant theory of the call-by-value lambda-calculus relies on weak evaluation and closed terms, that are natural hypotheses in the study of programming languages. To model proof assistants, however, strong evaluation and open terms are required, and it is well known that the operational semantics of call-by-value becomes problematic in this case. Here we study the intermediate setting—that we call Open Call-by-Value—of weak evaluation with open terms, on top of which Grégoire and Leroy designed the abstract machine of Coq. Various calculi for Open Call-by-Value already exist, each one with its pros and cons. This paper presents a detailed comparative study of the operational semantics of four of them, coming from different areas such as the study of abstract machines, denotational semantics, linear logic proof nets, and sequent calculus. We show that these calculi are all equivalent from a termination point of view, justifying the slogan Open Call-by-Value.

1 Introduction

Plotkin’s call-by-value λ -calculus [27] is at the heart of programming languages such as OCaml and proof assistants such as Coq. In the study of programming languages, call-by-value (CBV) evaluation is usually *weak*, *i.e.* it does not reduce under abstractions, and terms are assumed to be *closed*. These constraints give rise to a beautiful theory—let us call it *Closed CBV*—having the following *harmony property*, that relates rewriting and normal forms:

Closed normal forms are values (and values are normal forms)

where *values* are variables and abstractions. Harmony expresses a form of internal completeness with respect to unconstrained β -reduction: the restriction to CBV β -reduction (referred to as β_v -reduction, according to which a β -redex can be fired only when the argument is a value) has an impact on the order in which redexes are evaluated, but evaluation never gets stuck, as every β -redex will eventually become a β_v -redex and be fired, unless evaluation diverges.

It often happens, however, that one needs to go beyond the perfect setting of Closed CBV by considering *Strong CBV*, where reduction under abstractions is allowed and terms may be open, or the intermediate setting of *Open CBV*, where evaluation is weak but terms are not necessarily closed. The need arises, most notably, when trying to describe the implementation model of Coq [13],

but also from other motivations, as denotational semantics [26,29,3,8], monad and CPS translations and the associated equational theories [22,30,31,12,17], bisimulations [19], partial evaluation [18], linear logic proof nets [2], or cost models [1].

Naïve Open CBV. In call-by-name (CBN) turning to open terms or strong evaluation is harmless because CBN does not impose any special form to the arguments of β -redexes. On the contrary, turning to Open or Strong CBV is delicate. If one simply considers Plotkin’s weak β_v -reduction on open terms—let us call it *Naïve Open CBV*—then harmony does no longer hold, as there are open β -normal forms that are not values, *e.g.* xx , $x(\lambda y.y)$, $x(yz)$ or xyz . As a consequence, there are *stuck β -redexes* such as $(\lambda y.t)(xx)$, *i.e.* β -redexes that will never be fired because their argument is normal, but it is not a value, nor will it ever become one. Such stuck β -redexes are a disease typical of (Naïve) Open CBV, but they spread to Strong CBV as well (also in the closed case), because evaluating under abstraction forces to deal with locally open terms: *e.g.* the variable x is locally open with respect to $(\lambda y.t)(xx)$ in $s = \lambda x.((\lambda y.t)(xx))$.

The real issue with stuck β -redexes is that they prevent the creation of other redexes, and provide *premature β_v -normal forms*. The issue is serious, as it can affect termination, and thus impact on notions of observational equivalence. Let $\delta := \lambda x.(xx)$. The problem is exemplified by the terms t and u in Eq. (1) below.

$$t := ((\lambda y.\delta)(zz))\delta \qquad u := \delta((\lambda y.\delta)(zz)) \qquad (1)$$

In Naïve Open CBV, t and u are premature β_v -normal forms because they both have a stuck β -redex forbidding evaluation to keep going, while one would expect them to behave like the divergent term $\Omega := \delta\delta$ (see [26,29,3,2,8,15] and pp. 7-12).

Open CBV. In his seminal work, Plotkin already pointed out an asymmetry between CBN and CBV: his CPS translation is sound and complete for CBN, but only sound for CBV. This fact led to a number of studies about monad, CPS, and logical translations [22,30,31,21,12,17] that introduced many proposals of improved calculi for CBV. Starting with the seminal work of Paolini and Ronchi Della Rocca [26,24,29], the dissonance between open terms and CBV has been repeatedly pointed out and studied *per se* via various calculi [13,3,2,8,15,14,1]. A further point of view on CBV comes from the computational interpretation of sequent calculus due to Curien and Herbelin [9]. An important point is that the focus of most of these works is on Strong CBV.

These solutions inevitably extend β_v -reduction with some other rewriting rule(s) or constructor (as **let**-expressions) to deal with stuck β -redexes, or even go as far as changing the applicative structure of terms, as in the sequent calculus approach. They arise from different perspectives and each one has its pros and cons. By design, these calculi (when looked at in the context of Open CBV) are never observationally equivalent to Naïve Open CBV, as they all manage to (re)move stuck β -redexes and may diverge when Naïve Open CBV is instead stuck. Each one of these calculi, however, has its own notion of evaluation and normal form, and their mutual relationships are not evident.

The aim of this paper is to draw the attention of the community on Open CBV. We believe that it is somewhat deceiving that the mainstream operational theory of CBV, however elegant, has to rely on closed terms, because it restricts the modularity of the framework, and raises the suspicion that the true essence of CBV has yet to be found. There is a real gap, indeed, between Closed and Strong CBV, as Strong CBV cannot be seen as an iteration of Closed CBV under abstractions because such an iteration has to deal with open terms. To improve the implementation of Coq [13], Grégoire and Leroy see Strong CBV as the iteration of the intermediate case of Open CBV, but they do not explore its theory. Here we exalt their point of view, providing a thorough operational study of Open CBV. We insist on Open CBV rather than Strong CBV because:

1. Stuck β -redexes and premature β_v -normal forms already affect Open CBV;
2. Open CBV has a simpler rewriting theory than Strong CBV;
3. Our previous studies of Strong CBV in [3] and [8] naturally organized themselves as properties of Open CBV that were lifted to Strong CBV by a simple iteration under abstractions.

Our contributions are along two axes:

1. *Termination Equivalence of the Proposals*: we show that the proposed generalizations of Naïve Open CBV are all equivalent, in the sense that they have exactly the same sets of normalizing and diverging terms. So, *there is just one notion of Open CBV*, independently of its specific syntactic incarnation.
2. *Quantitative Analyses and Cost Models*: the termination results are complemented with quantitative analyses establishing precise relationships between the number of steps needed to evaluate a given term in the various calculi. In particular, we relate the cost models of the various proposals.

The Fab Four. We focus on four proposals for Open CBV, as other solutions, e.g. Moggi's [22] or Herbelin and Zimmerman's [17], are already known to be equivalent to these ones (see the end of Sect. 2):

1. *The Fireball Calculus* λ_{fire} , that extends values to *fireballs* by adding so-called *inert terms* in order to restore harmony—it was introduced without a name by Paolini and Ronchi Della Rocca [26,29], then rediscovered independently first by Leroy and Grégoire [13] to improve the implementation of Coq, and then by Accattoli and Sacerdoti Coen [1] to study cost models;
2. *The Value Substitution Calculus* λ_{vsub} , coming from the linear logic interpretation of CBV and using explicit substitutions and contextual rewriting rules to circumvent stuck β -redexes—it was introduced by Accattoli and Paolini [3] and it is a graph-free presentation of proof nets for the CBV λ -calculus [2];
3. *The Shuffling Calculus* λ_{shuf} , that has rules to shuffle constructors, similar to Regnier's σ -rules for CBN [28], as an alternative to explicit substitutions—it was introduced by Carraro and Guerrieri [8] (and further analyzed in [15,14]) to study the adequacy of Open/Strong CBV with respect to denotational semantics related to linear logic.

4. *The Value Sequent Calculus* λ_{vseq} , i.e. the intuitionistic fragment of Curien and Herbelin's $\bar{\lambda}\tilde{\mu}$ -calculus [9], that is a CBV calculus for classical logic providing a computational interpretation of sequent calculus rather than natural deduction (in turn a fragment of the $\bar{\lambda}\mu\tilde{\mu}$ -calculus [9], further studied in e.g. [5,10]).

A Robust Cost Model for Open CBV. The number of β_v -steps is the canonical time cost model of Closed CBV, as first proved by Blelloch and Greiner [7,32,11]. In [1], Accattoli and Sacerdoti Coen generalized this result: the number of steps in λ_{fire} is a reasonable cost model for Open CBV. Here we show that the number of steps in λ_{vsub} and λ_{vseq} are *linearly* related to the steps in λ_{fire} , thus providing reasonable cost models for these incarnations of Open CBV. As a consequence, complexity analyses can now be smoothly transferred between λ_{fire} , λ_{vsub} , and λ_{vseq} . Said differently, our results guarantee that the number of steps is a *robust* cost model for Open CBV, in the sense that it does not depend on the chosen incarnation. For λ_{shuf} we obtain a similar but strictly weaker result, due to some structural difficulties suggesting that λ_{shuf} is less apt to complexity analyses.

On the Value of The Paper. While the equivalences showed here are new, they might not be terribly surprising. Nonetheless, we think they are interesting, for the following reasons:

1. *Quantitative Relationships:* λ -calculi are usually related only *qualitatively*, while our relationships are *quantitative* and thus stronger: not only we show simulations, but we also relate the number of steps.
2. *Uniform View:* we provide a new uniform view on a known problem, that will hopefully avoid further proliferations of CBV calculi for open/strong settings.
3. *Expected but Non-Trivial:* while the equivalences are more or less expected, establishing them is informative, because it forces to reformulate and connect concepts among the different settings, and often tricky.
4. *Simple Rewriting Theory:* the relationships between the systems are developed using basic rewriting concepts. The technical development is simple, according to the best tradition of the CBV λ -calculus, and yet it provides a sharp and detailed decomposition of Open CBV evaluation.
5. *Connecting Different Worlds:* while λ_{fire} is related to Coq and implementations, λ_{vsub} and λ_{shuf} have a linear logic background, and λ_{vseq} is rooted in sequent calculus. With respect to linear logic, λ_{vsub} has been used for syntactical studies while λ_{shuf} for semantical ones. Our results therefore establish bridges between these different (sub)communities.

Finally, an essential contribution of this work is the recognition of Open CBV as a simple and yet rich framework in between Closed and Strong CBV.

Road Map. Sect. 2 provides an overview of the different presentations of Open CBV. Sect. 3 proves the termination equivalences for λ_{vsub} , λ_{fire} and λ_{shuf} , enriched with quantitative information. Sect. 4 proves the quantitative termination equivalence of λ_{vsub} and λ_{vseq} , via an intermediate calculus λ_{vsub_k} . Appendix A (p. 22) collects definitions and notations for the rewriting notions at work in the paper. Omitted proofs are in Appendix B (p. 23).

Terms	$t, u, s, r ::= v \mid tu$
Values	$v, v' ::= x \mid \lambda x.t$
Evaluation Contexts	$E ::= \langle \cdot \rangle \mid tE \mid Et$
RULE AT TOP LEVEL	CONTEXTUAL CLOSURE
$(\lambda x.t)\lambda y.u \mapsto_{\beta_\lambda} t\{x \leftarrow \lambda y.u\}$	$E\langle t \rangle \rightarrow_{\beta_\lambda} E\langle u \rangle \quad \text{if } t \mapsto_{\beta_\lambda} u$
$(\lambda x.t)y \mapsto_{\beta_y} t\{x \leftarrow y\}$	$E\langle t \rangle \rightarrow_{\beta_y} E\langle u \rangle \quad \text{if } t \mapsto_{\beta_y} u$
Reduction	$\rightarrow_{\beta_v} := \rightarrow_{\beta_\lambda} \cup \rightarrow_{\beta_y}$

Fig. 1. Naïve Open CBV λ_{Plot}

2 Incarnations of Open Call-by-Value

Here we recall Naïve Open CBV, noted λ_{Plot} , and introduce the four forms of Open CBV that will be compared (λ_{fire} , λ_{vsub} , λ_{shuf} , and λ_{vseq}) together with a semantic notion (*potential valuability*) reducing Open CBV to Closed CBV. In this paper terms are always possibly open. Moreover, we focus on Open CBV and avoid on purpose to study Strong CBV (we hint at how to define it, though).

Naïve Open CBV: Plotkin's calculus λ_{Plot} [27]. Naïve Open CBV is Plotkin's weak CBV λ -calculus λ_{Plot} on possibly open terms, defined in Fig. 1. Our presentation of the rewriting is unorthodox because we split β_v -reduction into two rules, according to the kind of value (abstraction or variable). The set of terms is denoted by Λ . Terms (in Λ) are always identified up to α -equivalence and the set of the free variables of a term t is denoted by $\text{fv}(t)$. We use $t\{x \leftarrow u\}$ for the term obtained by the capture-avoiding substitution of u for each free occurrence of x in t . Evaluation \rightarrow_{β_v} is weak and non-deterministic, since in the case of an application there is no fixed order in the evaluation of the left and right subterms. As it is well-known, non-determinism is only apparent: the system is strongly confluent (see Appendix A for a glossary of rewriting theory).

Proposition 1. \rightarrow_{β_y} , $\rightarrow_{\beta_\lambda}$ and \rightarrow_{β_v} are strongly confluent.

Proof p. 23

Strong confluence is a remarkable property, much stronger than plain confluence. It implies that, given a term, *all* derivations to its normal form (if any) have the *same length*, and that *normalization and strong normalization coincide*, *i.e.* if there is a normalizing derivation then there are no diverging derivations. Strong confluence will also hold for λ_{fire} , λ_{vsub} and λ_{vseq} , not for λ_{shuf} .

Let us come back to the splitting of \rightarrow_{β_v} . In Closed CBV it is well-known that \rightarrow_{β_y} is superfluous, at least as long as small-step evaluation is considered, see [4]. For Open CBV, \rightarrow_{β_y} is instead necessary, but—as we explained in the introduction—it is not enough, which is why we shall consider extensions of λ_{Plot} . The main problem of Naïve Open CBV is that there are stuck β -redexes breaking the harmony of the system. There are three kinds of solution: those *restoring a form of harmony* (λ_{fire}), to be thought as more semantical approaches; those *removing stuck β -redexes* (λ_{vsub} and λ_{shuf}), that are more syntactical in nature; those *changing the applicative structure of terms* (λ_{vseq}), inspired by sequent calculus.

Terms and Values	As in Plotkin's Open CBV (Fig. 1)
Fireballs	$f, f', f'' ::= \lambda x.t \mid i$
Inert Terms	$i, i', i'' ::= x f_1 \dots f_n \quad n \geq 0$
Evaluation Contexts	$E ::= \langle \cdot \rangle \mid tE \mid Et$
RULE AT TOP LEVEL	CONTEXTUAL CLOSURE
$(\lambda x.t)(\lambda y.u) \mapsto_{\beta_\lambda} t\{x \leftarrow \lambda y.u\}$	$E\langle t \rangle \rightarrow_{\beta_\lambda} E\langle u \rangle \quad \text{if } t \mapsto_{\beta_\lambda} u$
$(\lambda x.t)i \mapsto_{\beta_i} t\{x \leftarrow i\}$	$E\langle t \rangle \rightarrow_{\beta_i} E\langle u \rangle \quad \text{if } t \mapsto_{\beta_i} u$
Reduction	$\rightarrow_{\beta_f} := \rightarrow_{\beta_\lambda} \cup \rightarrow_{\beta_i}$

Fig. 2. The Fireball Calculus λ_{fire}

2.1 Open Call-by-Value 1: The Fireball Calculus λ_{fire}

The Fireball Calculus λ_{fire} , defined in Fig. 2, was introduced without a name by Paolini and Ronchi Della Rocca in [26] and [29, Def. 3.1.4, p. 36] where its basic properties are also proved. We give here a presentation inspired by Accattoli and Sacerdoti Coen's [1], departing from it only for inessential, cosmetic details. Terms, values and evaluation contexts are the same as in λ_{Plot} .

The idea is to restore harmony by generalizing \rightarrow_{β_y} to fire when the argument is a more general *inert term*—the new rule is noted \rightarrow_{β_i} . The generalization of values as to include inert terms is called *fireballs*. Actually fireballs and inert terms are defined by mutual induction (in Fig. 2). For instance, $\lambda x.y$ is a fireball as an abstraction, while x , $y(\lambda x.x)$, xy , and $(z(\lambda x.x))(zz)(\lambda y.(zy))$ are fireballs as inert terms. Note that ii' is an inert term for all inert terms i and i' . Inert terms can be equivalently defined as $i ::= x \mid if$ —such a definition is used in the proofs in the Appendix (where, moreover, inert terms that are not variables are referred to as *compound inert terms*). The main feature of an inert term is that it is open, normal and that when plugged in a context it cannot create a redex, hence the name (it is not a so-called *neutral term* because it might have redexes under abstractions). In Grégoire and Leroy's presentation [13], inert terms are called *accumulators* and fireballs are simply called values.

Evaluation is given by the fireball rule \rightarrow_{β_f} , that is the union of $\rightarrow_{\beta_\lambda}$ and \rightarrow_{β_i} . For instance, consider $t := ((\lambda y.\delta)(zz))\delta$ and $u := \delta((\lambda y.\delta)(zz))$ as in Eq. (1), p. 2: t and u are β_v -normal but they diverge when evaluated in λ_{fire} , as desired: $t \rightarrow_{\beta_i} \delta\delta \rightarrow_{\beta_\lambda} \delta\delta \rightarrow_{\beta_\lambda} \dots$ and $u \rightarrow_{\beta_i} \delta\delta \rightarrow_{\beta_\lambda} \delta\delta \rightarrow_{\beta_\lambda} \dots$.

The distinguished, key property of λ_{fire} is (for any $t \in \Lambda$):

Proposition 2 (Open Harmony). *t is β_f -normal iff t is a fireball.*

The advantage of λ_{fire} is its simple notion of normal form, *i.e.* fireballs, that have a clean syntactic description akin to that for call-by-name. The other calculi will lack a nice, natural notion of normal form. The drawback of the fireball calculus—and probably the reason why its importance did not emerge before—is the fact that as a strong calculus it is not confluent: this is due to the fact that fireballs are not closed by substitution (see [29, p. 37]). Indeed, if evaluation is strong, the following critical pair cannot be joined, where $t := (\lambda y.I)(\delta\delta)$ and

$I := \lambda z.z$ is the identity combinator:

$$I_{\beta\lambda\leftarrow} (\lambda x.I)\delta_{\beta_i\leftarrow} (\lambda x.(\lambda y.I)(xx))\delta \rightarrow_{\beta\lambda} t \rightarrow_{\beta\lambda} t \rightarrow_{\beta\lambda} \dots \quad (2)$$

On the other hand, as long as evaluation is weak (that is the case we consider) everything works fine—the strong case can then be caught by repeatedly iterating the weak one under abstraction, once a weak normal form has been obtained (thus forbidding the left part of (2)). In fact, the weak evaluation of λ_{fire} has a simple rewriting theory, as next proposition shows. In particular it is strongly confluent.

Proposition 3 (Basic Properties of λ_{fire}).

Proof p. 24

1. \rightarrow_{β_i} is strongly normalizing and strongly confluent.
2. $\rightarrow_{\beta_\lambda}$ and \rightarrow_{β_i} strongly commute.
3. \rightarrow_{β_f} is strongly confluent, and all β_f -normalizing derivations d from $t \in \Lambda$ (if any) have the same length $|d|_{\beta_f}$, the same number $|d|_{\beta_\lambda}$ of β_λ -steps, and the same number $|d|_{\beta_i}$ of β_i -steps.

2.2 Open Call-by-Value 2: The Value Substitution Calculus λ_{vsub}

Rewriting Preamble: Creations of Type 1 and 4. The problem with stuck β -redexes can be easily understood at the rewriting level as an issue about creations. According to Lévy [20], in the ordinary CBN λ -calculus redexes can be created in 3 ways. Creations of type 1 take the following form

$$((\lambda x.\lambda y.t)r)s \rightarrow_{\beta} (\lambda y.t\{x \leftarrow r\})s$$

where the redex involving λy and s has been created by the β -step. In Naïve Open CBV if r is a normal form and not a value then the creation cannot take place, blocking evaluation. This is the problem concerning the term t in Eq. (1), p. 2. In CBV there is another form of creation—of *type 4*—not considered by Lévy:

$$(\lambda x.t)((\lambda y.v)v') \rightarrow_{\beta_v} (\lambda x.t)(v\{y \leftarrow v'\})$$

i.e. a reduction in the argument turns the argument itself into a value, creating a β_v -redex. As before, in an open setting v' may be replaced by a normal form that is not a value, blocking the creation of type 4. This is exactly the problem concerning the term u in Eq. (1), p. 2.

The proposals of this and the next sections introduce some way to enable creations of type 1 and 4, without substituting stuck β -redexes nor inert terms.

The *value substitution calculus* λ_{vsub} of Accattoli and Paolini [3,2] was introduced as a calculus for Strong CBV inspired by linear logic proof nets. In Fig. 3 we present its adaptation to Open CBV, obtained by simply removing abstractions from evaluation contexts. It extends the syntax of terms with the constructor $[x \leftarrow u]$, called *explicit substitution* (shortened ES, to not be confused

vsub-Terms	$t, u, s ::= v \mid tu \mid t[x \leftarrow u]$
vsub-Values	$v ::= x \mid \lambda x. t$
Evaluation Contexts	$E ::= \langle \cdot \rangle \mid tE \mid Et \mid E[x \leftarrow u] \mid t[x \leftarrow E]$
Substitution Contexts	$L ::= \langle \cdot \rangle \mid L[x \leftarrow u]$
RULE AT TOP LEVEL	CONTEXTUAL CLOSURE
$L\langle \lambda x. t \rangle u \mapsto_m L\langle t[x \leftarrow u] \rangle$	$E\langle t \rangle \rightarrow_m E\langle u \rangle \quad \text{if } t \mapsto_m u$
$t[x \leftarrow L\langle \lambda y. u \rangle] \mapsto_{e_\lambda} L\langle t\{x \leftarrow \lambda y. u\} \rangle$	$E\langle t \rangle \rightarrow_{e_\lambda} E\langle u \rangle \quad \text{if } t \mapsto_{e_\lambda} u$
$t[x \leftarrow L\langle y \rangle] \mapsto_{e_y} L\langle t\{x \leftarrow y\} \rangle$	$E\langle t \rangle \rightarrow_{e_y} E\langle u \rangle \quad \text{if } t \mapsto_{e_y} u$
Reductions	$\rightarrow_e := \rightarrow_{e_\lambda} \cup \rightarrow_{e_y}, \quad \rightarrow_{\text{vsub}} := \rightarrow_m \cup \rightarrow_e$

Fig. 3. The Value Substitution Calculus λ_{vsub}

with the meta-level substitution $\{x \leftarrow u\}$). A **vsub**-term $t[x \leftarrow u]$ represents the delayed substitution of u for x in t , *i.e.* stands for **let** $x = u$ **in** t . So, $t[x \leftarrow u]$ binds the free occurrences of x in t . The set of **vsub**-terms—identified up to α -equivalence—is denoted by Λ_{vsub} (clearly $\Lambda \subsetneq \Lambda_{\text{vsub}}$).

ES are used to remove stuck β -redexes: the idea is that β -redexes can be fired whenever—even if the argument is not a (**vsub**-)value—by means of the *multiplicative rule* \rightarrow_m ; however the argument is not substituted but placed in a ES. The actual substitution is done only when the content of the ES is a **vsub**-value, by means of the *exponential rule* \rightarrow_e . These two rules are sometimes noted \rightarrow_{dB} (β at a distance) and \rightarrow_{vs} (substitution by value)—the names we use here are due to the interpretation of the calculus into linear logic proof-nets, see [2]. Note that in Fig. 3 the definition of the rewriting rules at top level \mapsto_m (resp. \mapsto_{e_y} and \mapsto_{e_λ}) assumes that the variables bound by the substitution context L are not free in u (resp. t). A characteristic feature coming from such an interpretation is that the rewriting rules are contextual, or *at a distance*: they are generalized as to act up to a list of substitutions (noted L , from *List*). Essentially, stuck β -redexes are turned into ES and then ignored by the rewriting rules—this is how creations of type 1 and 4 are enabled. For instance, the terms $t := ((\lambda y. \delta)(zz))\delta$ and $u := \delta((\lambda y. \delta)(zz))$ (as in Eq. (1), p. 2) are **e**-normal but $t \rightarrow_m \delta[y \leftarrow zz]\delta \rightarrow_m (xx)[x \leftarrow \delta][y \leftarrow zz] \rightarrow_e (\delta\delta)[y \leftarrow zz] \rightarrow_m (xx)[x \leftarrow \delta][y \leftarrow zz] \rightarrow_e (\delta\delta)[y \leftarrow zz] \rightarrow_m \dots$ and similarly for u .

The drawback of λ_{vsub} is that it requires explicit substitutions. The advantage of λ_{vsub} is its simple and well-behaved rewriting theory, even simpler than the rewriting for λ_{fire} , since every rule terminates separately (while β_λ does not)—in particular strong confluence holds. Moreover, the theory has a sort of flexible second level given by a notion of structural equivalence, coming up next.

Proof p. 26

Proposition 4 (Basic Properties of λ_{vsub} , [3]).

1. \rightarrow_m and \rightarrow_e are strongly normalizing and strongly confluent (separately).
2. \rightarrow_m and \rightarrow_e strongly commute.
3. $\rightarrow_{\text{vsub}}$ is strongly confluent, and all **vsub**-normalizing derivations d from $t \in \Lambda_{\text{vsub}}$ (if any) have the same length $|d|_{\text{vsub}}$, the same number $|d|_e$ of **e**-steps, and the same number $|d|_m$ of **m**-steps.
4. Let $t \in \Lambda$. For any **vsub**-derivation d from t , $|d|_e \leq |d|_m$.

Structural Equivalence. The theory of λ_{vsub} comes with a notion of structural equivalence \equiv , that equates **vsub**-terms that differ only for the position of ES. The basic idea is that the action of an ES via the exponential rule depends on the position of the ES itself only for inessential details (as long as the scope of binders is respected), namely the position of other ES, and thus can be abstracted away. A strong justification for the equivalence comes from the linear logic interpretation of λ_{vsub} , in which structurally equivalent **vsub**-terms translate to the same (recursively typed) proof net, see [2].

Structural equivalence \equiv is defined as the least equivalence relation on Λ_{vsub} closed by evaluation contexts (see Fig. 3) and generated by the following axioms:

$$\begin{array}{ll} t[y \leftarrow s][x \leftarrow u] \equiv_{\text{com}} t[x \leftarrow u][y \leftarrow s] & \text{if } y \notin \text{fv}(u) \text{ and } x \notin \text{fv}(s) \\ t s[x \leftarrow u] \equiv_{\text{@r}} (ts)[x \leftarrow u] & \text{if } x \notin \text{fv}(t) \\ t[x \leftarrow u] s \equiv_{\text{@l}} (ts)[x \leftarrow u] & \text{if } x \notin \text{fv}(s) \\ t[x \leftarrow u][y \leftarrow s] \equiv_{[\cdot]} t[x \leftarrow u][y \leftarrow s] & \text{if } y \notin \text{fv}(t) \end{array}$$

We set $\rightarrow_{\text{vsub}} := \equiv \rightarrow_{\text{vsub}} \equiv$ (i.e. for all $t, r \in \Lambda_{\text{vsub}}$: $t \rightarrow_{\text{vsub}} r$ iff $t \equiv u \rightarrow_{\text{vsub}} s \equiv r$ for some $u, s \in \Lambda_{\text{vsub}}$). The notation $\rightarrow_{\text{vsub}}^+$ keeps its usual meaning, while $\rightarrow_{\text{vsub}}^*$ stands for $\equiv \cup \rightarrow_{\text{vsub}}^+$, i.e. a **vsub**-derivation of length zero can apply \equiv and is not just the identity. As \equiv is reflexive, $\rightarrow_{\text{vsub}} \subseteq \rightarrow_{\text{vsub}}^*$.

The rewriting theory of λ_{vsub} enriched with structural equivalence \equiv is remarkably simple, as next lemma shows. In fact, \equiv commutes with evaluation, and can thus be postponed. Additionally, the commutation is *strong*, as it preserves the number and kind of steps—one says that it is a *strong bisimulation* (with respect to $\rightarrow_{\text{vsub}}$). In particular, the equivalence is not needed to compute and it does not break, or make more complex, any property of λ_{vsub} . On the contrary, it enhances the flexibility of the system: it will be essential to establish simple and clean relationships with the other calculi for Open CBV.

Lemma 5 (Basic Properties of Structural Equivalence \equiv , [3]). *Let $t, u \in \Lambda_{\text{vsub}}$ and $x \in \{\mathbf{m}, \mathbf{e}_\lambda, \mathbf{e}_y, \mathbf{e}, \text{vsub}\}$.*

1. Strong Bisimulation of \equiv wrt $\rightarrow_{\text{vsub}}$: *if $t \equiv u$ and $t \rightarrow_x t'$ then there exists $u' \in \Lambda_{\text{vsub}}$ such that $u \rightarrow_x u'$ and $t' \equiv u'$.*
2. Postponement of \equiv wrt $\rightarrow_{\text{vsub}}$: *if $d: t \rightarrow_{\text{vsub}}^* u$ then there are $s \equiv u$ and $e: t \rightarrow_{\text{vsub}}^* s$ such that $|d| = |e|$, $|d|_{\mathbf{e}_\lambda} = |e|_{\mathbf{e}_\lambda}$, $|d|_{\mathbf{e}_y} = |e|_{\mathbf{e}_y}$ and $|d|_{\mathbf{m}} = |e|_{\mathbf{m}}$.*
3. Normal Forms: *if $t \equiv u$ then t is x -normal iff u is x -normal.*
4. Strong confluence: $\rightarrow_{\text{vsub}} \equiv$ *is strongly confluent.*

2.3 Open Call-by-Value 3: The Shuffling Calculus λ_{shuf}

The calculus introduced by Carraro and Guerrieri in [8], and here deemed *Shuffling Calculus*, has the same syntax of terms as Plotkin's calculus. Two additional commutation rules help \rightarrow_{β_v} to deal with stuck β -redexes, by shuffling constructors so as to enable creations of type 1 and 4. As for λ_{vsub} , λ_{shuf} was actually introduced, and then used in [8,14,15], to study Strong CBV. In Fig. 4

Terms and Values	As in Plotkin's Open CBV (Fig. 1)
Balanced Contexts	$B ::= \langle \cdot \rangle \mid tB \mid Bt \mid (\lambda x.B)t$
RULE AT TOP LEVEL	CONTEXTUAL CLOSURE
$((\lambda x.t)u)s \mapsto_{\sigma_1} (\lambda x.ts)u, x \notin \text{fv}(s)$	$B\langle t \rangle \rightarrow_{\sigma_1^b} B\langle u \rangle \text{ if } t \mapsto_{\sigma_1} u$
$v((\lambda x.s)u) \mapsto_{\sigma_3} (\lambda x.vs)u, x \notin \text{fv}(v)$	$B\langle t \rangle \rightarrow_{\sigma_3^b} B\langle u \rangle \text{ if } t \mapsto_{\sigma_3} u$
$(\lambda x.t)v \mapsto_{\beta_v} t\{x \leftarrow v\}$	$B\langle t \rangle \rightarrow_{\beta_v^b} B\langle u \rangle \text{ if } t \mapsto_{\beta_v} u$
Reductions	$\rightarrow_{\sigma^b} := \rightarrow_{\sigma_1^b} \cup \rightarrow_{\sigma_3^b}, \quad \rightarrow_{\text{shuf}} := \rightarrow_{\beta_v^b} \cup \rightarrow_{\sigma^b}$

Fig. 4. The Shuffling Calculus λ_{shuf}

we present its adaptation to Open CBV, based on *balanced contexts*, a special notion of evaluation contexts. The reductions \rightarrow_{σ^b} and $\rightarrow_{\beta_v^b}$ are non-deterministic and—because of balanced contexts—can reduce under abstractions, but they are *morally* weak: they reduce under a λ only when the λ is applied to an argument. Note that the condition $x \notin \text{fv}(s)$ (resp. $x \notin \text{fv}(v)$) in the definition of the shuffling rule \mapsto_{σ_1} (resp. \mapsto_{σ_3}) can always be fulfilled by α -conversion.

The rewriting (shuffling) rules $\rightarrow_{\sigma_1^b}$ and $\rightarrow_{\sigma_3^b}$ unblock stuck β -redexes. For instance, consider the terms $t := ((\lambda y.\delta)(zz))\delta$ and $u := \delta((\lambda y.\delta)(zz))$ where $\delta := \lambda x.xx$ (as in Eq. (1), p. 2): t and u are β_v^b -normal but $t \rightarrow_{\sigma_1^b} (\lambda y.\delta\delta)(zz) \rightarrow_{\beta_v^b} (\lambda y.\delta\delta)(zz) \rightarrow_{\beta_v^b} \dots$ and $u \rightarrow_{\sigma_3^b} (\lambda y.\delta\delta)(zz) \rightarrow_{\beta_v^b} (\lambda x.\delta\delta)(zz) \rightarrow_{\beta_v^b} \dots$.

The similar shuffling rules in CBN, better known as Regnier's σ -rules [28], are *contained* in CBN β -equivalence, while in Open (and Strong) CBV they are more interesting because they are not contained into (*i.e.* they enrich) β_v -equivalence.

The advantage of λ_{shuf} is with respect to denotational investigations. In [8], λ_{shuf} is indeed used to prove various semantical results in connection to linear logic, resource calculi, and the notion of Taylor expansion due to Ehrhard. In particular, in [8] it has been proved the adequacy of λ_{shuf} with respect to the relational model induced by linear logic: a by-product of our paper is the extension of this adequacy result to all incarnations of Open CBV. The drawback of λ_{shuf} is its technical rewriting theory. We summarize some properties of λ_{shuf} :

Proposition 6 (Basic Properties of λ_{shuf} , [8]).

1. Let $t, u, s \in \Lambda$. If $t \rightarrow_{\beta_v^b} u$ and $t \rightarrow_{\sigma^b} s$ then $u \neq s$.
2. \rightarrow_{σ^b} is strongly normalizing and (not strongly) confluent.
3. $\rightarrow_{\text{shuf}}$ is (not strongly) confluent.
4. Let $t \in \Lambda$: t is strongly shuf-normalizable iff t is shuf-normalizable.

In contrast to λ_{fire} and λ_{vsub} , λ_{shuf} is not strongly confluent and not all shuf-normalizing derivations (if any) from a given term have the same length (consider, for instance, all shuf-normalizing derivations from $(\lambda y.z)(\delta(zz))\delta$). Nonetheless, normalization and strong normalization still coincide in λ_{shuf} (Prop. 6.4), and Cor. 18 in Sect. 3 will show that the discrepancy is encapsulated inside the additional shuffling rules, since all shuf-normalizing derivations (if any) from a given term have the same number of β_v^b -steps.

Commands	$c, c' ::= \langle v e \rangle$
Values	$v, v' ::= x \mid \lambda x.c$
Environments	$e, e' ::= \epsilon \mid \tilde{\mu}x.c \mid v \cdot e$
Command Evaluation Contexts	$C ::= \langle \cdot \rangle \mid D\langle \tilde{\mu}x.C \rangle$
Environment Evaluation Contexts	$D ::= \langle v \mid \langle \cdot \rangle \rangle \mid D\langle v \cdot \langle \cdot \rangle \rangle$
RULE AT TOP LEVEL	CONTEXTUAL CLOSURE
$\langle \lambda x.c \mid v \cdot e \rangle \mapsto_{\tilde{\lambda}} \langle v \mid (\tilde{\mu}x.c)@e \rangle$	$C\langle c \rangle \rightarrow_{\tilde{\lambda}} C\langle c' \rangle \quad \text{if } c \mapsto_{\tilde{\lambda}} c'$
$\langle v \mid \tilde{\mu}x.c \rangle \mapsto_{\tilde{\mu}} c\{x \leftarrow v\}$	$C\langle c \rangle \rightarrow_{\tilde{\mu}} C\langle c' \rangle \quad \text{if } c \mapsto_{\tilde{\mu}} c'$
Reduction	$\rightarrow_{vseq} := \rightarrow_{\tilde{\lambda}} \cup \rightarrow_{\tilde{\mu}}$

Fig. 5. The Value Sequent Calculus λ_{vseq}

2.4 Open Call-by-Value 4: The Value Sequent Calculus λ_{vseq}

A more radical approach to the removal of stuck β -redexes is provided by what is here called the *Value Sequent Calculus* λ_{vseq} , defined in Fig. 5. In λ_{vseq} , it is the applicative structure of terms that is altered, by replacing the application constructor with more constructs, namely commands c and environments e . Morally, λ_{vseq} looks at a sequence of applications from the head, that is the value on the left of a command $\langle v | e \rangle$ rather than from the tail as in natural deduction. In fact, λ_{vseq} is a handy presentation of the intuitionistic fragment of $\bar{\lambda}\tilde{\mu}$, that in turn is the CBV fragment of $\bar{\lambda}\mu\tilde{\mu}$, a calculus obtained as the computational interpretation of a sequent calculus for classical logic. Both $\bar{\lambda}\tilde{\mu}$ and $\bar{\lambda}\mu\tilde{\mu}$ are due to Curien and Herbelin [9], see [5,10] for further investigations about these systems.

A peculiar trait of the sequent calculus approach is the environment constructor $\tilde{\mu}x.c$, that is a binder for the free occurrences of x in c . It is often said that it is a sort of explicit substitution—we will see exactly in which sense, in Sect. 4.

The change of the intuitionistic variant λ_{vseq} with respect to $\bar{\lambda}\tilde{\mu}$ is that λ_{vseq} does not need the syntactic category of co-variables α , as there can be only one of them, denoted here by ϵ . From a logical viewpoint, this is due to the fact that in intuitionistic sequent calculus the right-hand-side of \vdash has exactly one formula, that is neither contraction nor weakening are allowed on the right. Consequently, the binary abstraction $\lambda(x, \alpha).c$ of $\bar{\lambda}\tilde{\mu}$ is replaced by a more traditional unary one $\lambda x.c$, and substitution on co-variables is replaced by a notion of *appending of environments*, defined by mutual induction on commands and environments as follows:

$$\begin{aligned}
\langle v | e' \rangle @ e &:= \langle v | e' @ e \rangle & \epsilon @ e &:= e \\
(v \cdot e') @ e &:= v \cdot (e' @ e) & (\tilde{\mu}x.c) @ e &:= \tilde{\mu}y.(c\{x \leftarrow y\} @ e) \text{ with } y \notin \text{fv}(c) \cup \text{fv}(e)
\end{aligned}$$

Essentially, $c @ e$ is a capture-avoiding substitution of e for the only occurrence of ϵ in c that is out of all abstractions, standing for the output of the term. The append operation is used in the rewrite rule $\rightarrow_{\tilde{\lambda}}$ of λ_{vseq} (Fig. 5). Strong CBV can be obtained by simply extending the grammar of evaluation contexts to commands under abstractions.

We will provide a translation from λ_{vsub} to λ_{vseq} that, beyond termination equivalence, will show that switching to a sequent calculus representation is equivalent to a transformation in administrative normal form [30].

The advantage of λ_{vseq} is that it avoids both rules at a distance and shuffling rules. The drawback of λ_{vseq} is that, syntactically, it requires to step out of the λ -calculus. We will show in Sect. 4 how to reformulate it as a fragment of λ_{vsub} , *i.e.* in natural deduction. However, it will still be necessary to restrict the application constructor, thus preventing the natural way of writing terms.

The rewriting of λ_{vseq} is very well-behaved, in particular it is strongly confluent and every rewriting rule terminates separately.

Proof p. 34

Proposition 7 (Basic properties of λ_{vseq}).

1. $\rightarrow_{\bar{\lambda}}$ and $\rightarrow_{\bar{\mu}}$ are strongly normalizing and strongly confluent (separately).
2. $\rightarrow_{\bar{\lambda}}$ and $\rightarrow_{\bar{\mu}}$ strongly commute.
3. $\rightarrow_{\text{vseq}}$ is strongly confluent, and all **vseq**-normalizing derivations d from a command c (if any) have the same length $|d|$, the same number $|d|_{\bar{\mu}}$ of $\bar{\mu}$ -steps, and the same number $|d|_{\bar{\lambda}}$ of $\bar{\lambda}$ -steps.

2.5 Variations on a Theme

Reducing Open to Closed Call-by-Value: Potential Valuability. Potential valuability relates Naïve Open CBV to Closed CBV via a meta-level substitution closing open terms: a (possibly open) term t is *potentially valuable* if there is a substitution of (closed) *values* for its free variables, for which it β_v -evaluates to a (closed) *value*.³ In Naïve Open CBV, potentially valuable terms do not coincide with normalizable terms because of premature β_v -normal forms—such as t and u in Eq. (1) at p. 2— which are not potentially valuable.

Paolini, Ronchi Della Rocca and, later, Pimentel [26,24,29,25,23] gave several operational, logical, and semantical characterizations of potentially valuable terms in Naïve Open CBV. In particular, in [26,29] it is proved that a term is potentially valuable in Plotkin’s Naïve Open CBV iff its normalizable in λ_{fire} .

Potentially valuable terms can be defined for every incarnation of Open CBV: it is enough to update the notions of evaluation and values in the above definition to the considered calculus. This has been done for λ_{shuf} in [8], and for λ_{vsub} in [3]. For both calculi it has been proved that, in the weak setting, potentially valuable terms coincide with normalizable terms. In [15], it has been proved that Plotkin’s potentially valuable terms coincide with **shuf**-potentially valuable terms (which coincide in turn with **shuf**-normalizable terms). Our paper makes a further step: proving that termination coincides for λ_{fire} , λ_{vsub} , λ_{shuf} , and λ_{vseq} it implies that all their notions of potential valuability coincide with Plotkin’s, *i.e.* there is just one notion of potential valuability for Open (and Strong) CBV.

Open CBV 5, 6, 7, ... The literature contains many other calculi for CBV, usually presented for Strong CBV and easily adaptable to Open CBV. Some of them have **let**-expressions (avatars of ES) and all of them have rules permuting constructors, therefore they lie somewhere in between λ_{vsub} and λ_{shuf} . Often, they

³ Potential valuability for Plotkin’s CBV λ -calculus can be equivalently defined using weak or strong β_v -reduction: it is the same notion for Naïve Open and Strong CBV.

have been developed for other purposes, usually to investigate the relationship with monad or CPS translations. Moggi's equational theory [22] is a classic standard of reference, known to coincide with that of Sabry and Felleisen [30], Sabry and Wadler [31], Dychoff and Lengrand [12], Herbelin and Zimmerman [17] and Maraist et al's λ_{let} in [21]. In [3], λ_{vsub} modulo \equiv is shown to be termination equivalent to Herbelin and Zimmerman's calculus, and to strictly contain its equational theory, and thus Moggi's. At the level of rewriting these presentations of Open CBV are all more involved than those that we consider here. Their equivalence to our calculi can be shown along the lines of that of λ_{shuf} with λ_{vsub} .

3 Quantitative Equivalence of λ_{fire} , λ_{vsub} , and λ_{shuf}

Here we show the equivalence with respect to termination of λ_{fire} , λ_{vsub} , and λ_{shuf} , enriched with quantitative information on the number of steps.

On the Proof Technique. We show that termination in λ_{vsub} implies termination in λ_{fire} and λ_{shuf} by studying simulations of λ_{fire} and λ_{shuf} into λ_{vsub} . To prove the converse implications we do not use inverse simulations. Alternatively, we show that β_f - and $shuf$ -normal forms are essentially projected into $vsub$ -normal forms, so that if evaluation terminates in λ_{fire} or λ_{shuf} then it also terminates on λ_{vsub} .

Such a simple technique works because in the systems under study *normalization and strong normalization coincide*: if there is a normalizing derivation from a given term t then there are no diverging derivations from t (for λ_{vsub} and λ_{fire} it follows from strong confluence, for λ_{shuf} is given by Prop. 6.4). This fact is also the reason why the statements of our equivalences (forthcoming Cor. 13 and Cor. 17) address a single derivation from t rather than considering *all* derivations from t . Moreover, for any calculus, all normalizing derivations from t have the same number of steps (in λ_{shuf} it holds for β_v^b -steps, see Cor. 18), hence also the quantitative claims of Cor. 13 and Cor. 17 hold actually for *all* normalizing derivations from t .

In both simulations, the structural equivalence \equiv of λ_{vsub} plays a role.

3.1 Equivalence of λ_{fire} and λ_{vsub}

A single β_v -step $(\lambda x.t)v \rightarrow_{\beta_v} t\{x \leftarrow v\}$ is simulated in λ_{vsub} by two steps: $(\lambda x.t)v \rightarrow_m t[x \leftarrow v] \rightarrow_e t\{x \leftarrow v\}$, *i.e.* a m -step that creates a ES, and a e -step that turns the ES into the meta-level substitution performed by the β_v -step. The simulation of an inert step of λ_{fire} is instead trickier, because in λ_{vsub} there is no rule to substitute an inert term, if it is not a variable. The idea is that an inert step $(\lambda x.t)i \rightarrow_{\beta_i} t\{x \leftarrow i\}$ is simulated only by $(\lambda x.t)i \rightarrow_m t[x \leftarrow i]$, *i.e.* only by the m -step that creates the ES, and such a ES will never be fired—so the simulation is up to the unfolding of substitutions containing inert terms (defined right next).

Everything works because of the key property of inert terms: they are normal and their substitution cannot create redexes, so it is useless to substitute them.

The *unfolding* of a **vsub**-term t is the term $t\downarrow$ obtained from t by turning ES into meta-level substitutions; it is defined by:

$$x\downarrow := x \quad (tu)\downarrow := t\downarrow u\downarrow \quad (\lambda x.t)\downarrow := \lambda x.t\downarrow \quad (t[x \leftarrow u])\downarrow := t\downarrow\{x \leftarrow u\downarrow\}$$

For all $t, u \in \Lambda_{\text{vsub}}$, $t \equiv u$ implies $t\downarrow = u\downarrow$. Also, $t\downarrow = t$ iff $t \in \Lambda$.

In the simulation we are going to show, structural equivalence \equiv plays a role. It is used to *clean* the **vsub**-terms (with ES) obtained by simulation, putting them in a canonical form where ES do not appear among other constructors.

A **vsub**-term is *clean* if it has the form $u[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$ (with $n \in \mathbb{N}$), $u \in \Lambda$ is called the *body*, and $i_1, \dots, i_n \in \Lambda$ are inert terms. Clearly, any term (as it is without ES) is clean. We first show how to simulate a single fireball step.

Proof p. 36

Lemma 8 (Simulation of a β_f -Step in λ_{vsub}). *Let $t, u \in \Lambda$.*

1. *If $t \rightarrow_{\beta_\lambda} u$ then $t \rightarrow_{\mathbf{m}} \rightarrow_{\mathbf{e}_\lambda} u$.*
2. *If $t \rightarrow_{\beta_i} u$ then $t \rightarrow_{\mathbf{m}} \equiv s$, with $s \in \Lambda_{\text{vsub}}$ clean and $s\downarrow = u$.*

We cannot simulate derivations by iterating Lemma 8, because the starting term t has no ES but the simulation of inert steps introduces ES. Hence, we have to generalize Lemma 8 up to the unfolding of ES. In general, unfolding ES is a dangerous operation with respect to (non-)termination, as it may erase a diverging subterm (e.g. $t := x[y \leftarrow \delta\delta]$ is **vsub**-divergent and $t\downarrow = x$ is normal). In our case, however, the simulation produces clean **vsub**-terms, so the unfolding is safe since it can erase only inert terms and cannot create, erase, nor carry redexes.

By means of a technical lemma in the appendix we obtain:

Proof p. 37

Lemma 9 (Projection of a β_f -Step on $\rightarrow_{\text{vsub}}$ via Unfolding). *Let t be a clean **vsub**-term and u be a term.*

1. *If $t\downarrow \rightarrow_{\beta_\lambda} u$ then $t \rightarrow_{\mathbf{m}} \rightarrow_{\mathbf{e}_\lambda} s$, with $s \in \Lambda_{\text{vsub}}$ clean and $s\downarrow = u$.*
2. *If $t\downarrow \rightarrow_{\beta_i} u$ then $t \rightarrow_{\mathbf{m}} \equiv s$, with $s \in \Lambda_{\text{vsub}}$ clean and $s\downarrow = u$.*

Via Lemma 9 we can now simulate whole derivations (in forthcoming Thm. 12).

Simulation and Normal Forms. The next step towards the equivalence is to relate normal forms in λ_{fire} (aka fireballs) to those in λ_{vsub} . The relationship is not perfect, since the simulation does not directly map the former to the latter—we have to work a little bit more. First of all, let us characterize the terms in λ_{vsub} obtained by projecting normalizing derivations (that always produce a fireball).

Proof p. 38

Lemma 10. *Let t be a clean **vsub**-term. If $t\downarrow$ is a fireball, then t is $\{\mathbf{m}, \mathbf{e}_\lambda\}$ -normal and its body is a fireball.*

Now, a $\{\mathbf{m}, \mathbf{e}_\lambda\}$ -normal form t morally is **vsub**-normal, as $\rightarrow_{\mathbf{e}_y}$ terminates (Prop. 4.1) and it cannot create $\{\mathbf{m}, \mathbf{e}_\lambda\}$ -redexes. The part about creations is better expressed as a postponement property.

Proof p. 38

Lemma 11 (Linear Postponement of \rightarrow_{e_y}). *Let $t, u \in \Lambda_{\text{vsub}}$. If $d: t \rightarrow_{\text{vsub}}^* u$ then $e: t \rightarrow_{\mathfrak{m}, e_\lambda}^* \rightarrow_{e_y}^* u$ with $|e|_{\text{vsub}} = |d|_{\text{vsub}}$, $|e|_{\mathfrak{m}} = |d|_{\mathfrak{m}}$, $|e|_e = |d|_e$ and $|e|_{e_\lambda} \geq |d|_{e_\lambda}$.*

The next theorem puts all the pieces together.

Theorem 12 (Quantitative Simulation of λ_{fire} in λ_{vsub}). *Let $t, u \in \Lambda$. If $d: t \rightarrow_{\beta_f}^* u$ then there are $s, r \in \Lambda_{\text{vsub}}$ and $e: t \rightarrow_{\text{vsub}}^* r$ such that*

Proof p. 41

1. Qualitative Relationship: $r \equiv s$, $u = s \downarrow = r \downarrow$ and s is clean;
2. Quantitative Relationship:
 1. Multiplicative Steps: $|d|_{\beta_f} = |e|_{\mathfrak{m}}$;
 2. Exponential (Abstraction) Steps: $|d|_{\beta_\lambda} = |e|_{e_\lambda} = |e|_e$.
3. Normal Forms: if u is β_f -normal then there exists $g: r \rightarrow_{e_y}^* q$ such that q is a vsub -normal form and $|g|_{e_y} \leq |e|_{\mathfrak{m}} - |e|_{e_\lambda}$.

Corollary 13 (Linear Termination Equivalence of λ_{vsub} and λ_{fire}). *Let $t \in \Lambda$. There is a β_f -normalizing derivation d from t iff there is a vsub -normalizing derivation e from t . Moreover, $|d|_{\beta_f} \leq |e|_{\text{vsub}} \leq 2|d|_{\beta_f}$, i.e. they are linearly related.*

Proof p. 42

The number of β_f -steps in λ_{fire} is a reasonable cost model for Open CBV [1]. Our result implies that also the number of \mathfrak{m} -steps in λ_{vsub} is a reasonable cost model, since the number of \mathfrak{m} -steps is exactly the number of β_f -steps. This fact is quite surprising: in λ_{fire} arguments of β_f -redexes are required to be fireballs, while for \mathfrak{m} -redexes there are no restrictions on arguments, and yet in any normalizing derivation their number coincide. Note, moreover, that e -steps are linear in \mathfrak{m} -steps, but only because the initial term has no ES: in general, this is not true.

3.2 Equivalence of λ_{shuf} and λ_{vsub}

A derivation $d: t \rightarrow_{\text{shuf}}^* u$ in λ_{shuf} is simulated via a projection on multiplicative normal forms in λ_{vsub} , i.e. as a derivation $\mathfrak{m}(t) \rightarrow_{\text{vsub}}^* \mathfrak{m}(u)$ (for any vsub -term t , its multiplicative and exponential normal forms, denoted by $\mathfrak{m}(t)$ and $\mathfrak{e}(t)$ respectively, exist and are unique by Prop. 4). Indeed, a β_v^b -step of λ_{shuf} is simulated in λ_{vsub} by a e -step followed by some \mathfrak{m} -steps to reach the \mathfrak{m} -normal form. Shuffling rules \rightarrow_{σ^b} of λ_{shuf} are simulated by structural equivalence \equiv in λ_{vsub} : applying $\mathfrak{m}(\cdot)$ to $((\lambda x.t)u)s \rightarrow_{\sigma_1^b} (\lambda x.(ts))u$ we obtain exactly an instance of the axiom \equiv_{a1} defining \equiv : $\mathfrak{m}(t)[x \leftarrow \mathfrak{m}(u)]\mathfrak{m}(s) \equiv_{\text{a1}} (\mathfrak{m}(t)\mathfrak{m}(s))[x \leftarrow \mathfrak{m}(u)]$ (with the side conditions matching exactly). Similarly, $\rightarrow_{\sigma_3^b}$ projects to \equiv_{a2} or \equiv_{a3} (depending on whether v in $\rightarrow_{\sigma_3^b}$ is a variable or an abstraction). Therefore,

Lemma 14 (Projecting a shuf-Step on $\rightarrow_{\text{vsub}}$ via \mathfrak{m} -NF). *Let $t, u \in \Lambda$.*

Proof p. 43

1. If $t \rightarrow_{\sigma^b} u$ then $\mathfrak{m}(t) \equiv \mathfrak{m}(u)$.
2. If $t \rightarrow_{\beta_v^b} u$ then $\mathfrak{m}(t) \rightarrow_e \rightarrow_{\mathfrak{m}}^* \mathfrak{m}(u)$.

In contrast to the simulation of λ_{fire} in λ_{vsub} , here the projection of a single step can be extended to derivations without problems, obtaining that the number

of β_v^b -steps in λ_{shuf} matches exactly the number of **e**-steps in λ_{vsub} . Additionally, we apply the postponement of \equiv (Lemma 5.2), factoring out the use of \equiv (*i.e.* of shuffling rules) without affecting the number of **e**-steps.

To obtain the termination equivalence we also need to study normal forms. Luckily, the case of λ_{shuf} is simpler than that of λ_{fire} , as next lemma shows.

Proof p. 43

Lemma 15 (Projection Preserves Normal Forms). *Let $t \in \Lambda$. If t is shuf-normal then $\mathfrak{m}(t)$ is vsub-normal.*

The next theorem puts all the pieces together (for any shuf-derivation d , $|d|_{\beta_v^b}$ is the number of β_v^b -steps in d : this notion is well defined by Prop. 6.1).

Proof p. 44

Theorem 16 (Quantitative Simulation of λ_{shuf} in λ_{vsub}). *Let $t, u \in \Lambda$. If $d: t \rightarrow_{\text{shuf}}^* u$ then there are $s \in \Lambda_{\text{vsub}}$ and $e: t \rightarrow_{\text{vsub}}^* s$ such that*

1. Qualitative Relationship: $s \equiv \mathfrak{m}(u)$;
2. Quantitative Relationship (Exponential Steps): $|d|_{\beta_v^b} = |e|_{\mathfrak{e}}$;
3. Normal Form: *if u is shuf-normal then s and $\mathfrak{m}(u)$ are vsub-normal.*

Proof p. 45

Corollary 17 (Termination Equivalence of λ_{vsub} and λ_{shuf}). *Let $t \in \Lambda$. There is a shuf-normalizing derivation d from t iff there is a vsub-normalizing derivation e from t . Moreover, $|d|_{\beta_v^b} = |e|_{\mathfrak{e}}$.*

The obtained quantitative equivalence has an interesting corollary that shows some light on why λ_{shuf} is not strongly confluent. Our simulation maps β_v^b -steps in λ_{shuf} to exponential steps in λ_{vsub} , that are strongly confluent, and thus in equal number in all normalizing derivations (if any) from a given term. Therefore,

Proof p. 45

Corollary 18 (Number of β_v^b -Steps is Invariant). *All shuf-normalizing derivations from $t \in \Lambda$ (if any) have the same number of β_v^b -steps.*

Said differently, in λ_{shuf} normalizing derivations may have different lengths but the difference is encapsulated inside the shuffling rules $\rightarrow_{\sigma_1^b}$ and $\rightarrow_{\sigma_3^b}$.

Concerning the cost model, things are subtler for λ_{shuf} . Note that the relationship between λ_{shuf} and λ_{vsub} uses the number of **e**-steps, while the cost model (inherited from λ_{fire}) is the number of **m**-steps. Do **e**-steps provide a reasonable cost model? Probably not, because there is a family of terms that evaluate in exponentially more **m**-steps than **e**-steps. Details are left to a longer version.

4 Quantitative Equivalence of λ_{vsub} and λ_{vseq} , via λ_{vsub_k}

The quantitative termination equivalence of λ_{vsub} and λ_{vseq} is shown in two steps: first, we identify a sub-calculus λ_{vsub_k} of λ_{vsub} equivalent to the whole of λ_{vsub} , and then show that λ_{vsub_k} and λ_{vseq} are equivalent (actually isomorphic). Both steps reuse the technique of Sect. 3, *i.e.* simulation plus study of normal forms.

4.1 Equivalence of λ_{vsub_k} and λ_{vsub}

The kernel λ_{vsub_k} of λ_{vsub} is the sublanguage of λ_{vsub} obtained by replacing the application constructor tu with the restricted form tv where the right subterm can only be a value v —i.e., λ_{vsub_k} is the language of so-called *administrative normal forms* [30] of λ_{vsub} . The rewriting rules are the same of λ_{vsub} . It is easy to see that λ_{vsub_k} is stable by vsub -reduction. For lack of space, more details about λ_{vsub_k} have been moved to Appendix B.3 (page 45).

The translation $(\cdot)^+$ of λ_{vsub} into λ_{vsub_k} , which simply places the argument of an application into an ES, is defined by (note that $\mathbf{fv}(t) = \mathbf{fv}(t^+)$ for all $t \in \Lambda_{\text{vsub}}$):

$$\begin{aligned} x^+ &:= x & (tu)^+ &:= (t^+x)[x \leftarrow u^+] \text{ where } x \text{ is fresh} \\ (\lambda x.t)^+ &:= \lambda x.t^+ & t[x \leftarrow u]^+ &:= t^+[x \leftarrow u^+] \end{aligned}$$

Lemma 19 (Simulation). *Let $t, u \in \Lambda_{\text{vsub}}$.*

Proof p. 46

1. Multiplicative: *if $t \rightarrow_{\mathbf{m}} u$ then $t^+ \rightarrow_{\mathbf{m}} \rightarrow_{\mathbf{e}_y} u^+ \equiv u^+$;*
2. Exponential: *if $t \rightarrow_{\mathbf{e}_\lambda} u$ then $t^+ \rightarrow_{\mathbf{e}_\lambda} u^+$, and if $t \rightarrow_{\mathbf{e}_y} u$ then $t^+ \rightarrow_{\mathbf{e}_y} u^+$.*
3. Structural Equivalence: *$t \equiv u$ implies $t^+ \equiv u^+$.*

The translation of a vsub -normal form is not vsub_k -normal (e.g. $(xy)^+ = (xz)[z \leftarrow y]$) but a further exponential normalization provides a vsub_k -normal form.

Theorem 20 (Quantitative Simulation of λ_{vsub} in λ_{vsub_k}). *Let $t, u \in \Lambda_{\text{vsub}}$. If $d: t \rightarrow_{\text{vsub}}^* u$ then there are $s \in \Lambda_{\text{vsub}_k}$ and $e: t^+ \rightarrow_{\text{vsub}_k}^* s$ such that*

Proof p. 48

1. Qualitative Relationship: $s \equiv u^+$;
2. Quantitative Relationship:
 1. Multiplicative Steps: $|e|_{\mathbf{m}} = |d|_{\mathbf{m}}$;
 2. Exponential Steps: $|e|_{\mathbf{e}_\lambda} = |d|_{\mathbf{e}_\lambda}$ and $|e|_{\mathbf{e}_y} = |d|_{\mathbf{e}_y} + |d|_{\mathbf{m}}$;
3. Normal Form: *if u is vsub -normal then s is \mathbf{m} -normal and $\mathbf{e}(s)$ is vsub_k -normal.*

Unfortunately, the length of the exponential normalization in Thm. 20.3 cannot be easily bounded, forbidding a precise quantitative equivalence. Note however that turning from λ_{vsub} to its kernel λ_{vsub_k} does not change the number of multiplicative steps: the transformation preserves the cost model.

Corollary 21 (Termination and Cost Equivalence of λ_{vsub} and λ_{vsub_k}). *Let $t \in \Lambda_{\text{vsub}}$. There exists a vsub -normalizing derivation d from t iff there exists a vsub_k -normalizing derivation e from t^+ . Moreover, $|d|_{\mathbf{m}} = |e|_{\mathbf{m}}$.*

Proof p. 48

4.2 Equivalence of λ_{vsub_k} and λ_{vseq}

The translation $\underline{\cdot}$ of λ_{vsub_k} into λ_{vseq} relies on an auxiliary translation $(\cdot)^\bullet$ of values and it is defined as follows:

$$\begin{aligned} x^\bullet &:= x & (\lambda x.t)^\bullet &:= \lambda x.\underline{t} \\ \underline{v} &:= \langle v | \epsilon \rangle & \underline{tv} &:= \underline{t} @ (v^\bullet \cdot \epsilon) & \underline{t[x \leftarrow u]} &:= \underline{u} @ \tilde{\mu} x.\underline{t} \end{aligned}$$

Note the subtle mapping of ES to $\tilde{\mu}$: ES correspond to appendings of $\tilde{\mu}$ to the output of the term u to be substituted, and not of the term t where to substitute.

It is not hard to see that λ_{vsub_k} and λ_{vseq} are actually isomorphic, where the converse translation $(\cdot)^\diamond$, that maps values and commands to terms, and environments to evaluation contexts, is given by:

$$\begin{aligned} x^\diamond &:= x & \epsilon^\diamond &:= \langle \cdot \rangle & \langle v | e \rangle^\diamond &:= e^\diamond \langle v^\diamond \rangle \\ (\lambda x.c)^\diamond &:= \lambda x.c^\diamond & (v \cdot e)^\diamond &:= e^\diamond \langle \langle \cdot \rangle v^\diamond \rangle & (\tilde{\mu}x.c)^\diamond &:= c^\diamond [x \leftarrow \langle \cdot \rangle] \end{aligned}$$

For the sake of uniformity, we follow the same structure of the other weaker equivalences (*i.e.* simulation plus mapping of normal forms, here working smoothly) rather than proving the isomorphism formally. The simulation maps multiplicative steps to $\bar{\lambda}$ steps, whose number, then, is a reasonable cost model for λ_{vseq} .

Proof p. 50

Lemma 22 (Simulation of $\rightarrow_{\text{vsub}_k}$ by $\rightarrow_{\text{vseq}}$). *Let t and u be vsub_k -terms.*

1. Multiplicative: *if $t \rightarrow_m u$ then $\underline{t} \rightarrow_{\bar{\lambda}} \underline{u}$.*
2. Exponential: *if $t \rightarrow_e u$ then $\underline{t} \rightarrow_{\tilde{\mu}} \underline{u}$.*

Proof p. 51

Theorem 23 (Quantitative Simulation of λ_{vsub_k} in λ_{vseq}). *Let t and u be vsub_k -terms. If $d: t \rightarrow_{\text{vsub}_k}^* u$ then there is $e: \underline{t} \rightarrow_{\text{vseq}}^* \underline{u}$ such that*

1. Multiplicative Steps: $|d|_m = |e|_{\bar{\lambda}}$ (the number $\bar{\lambda}$ -steps in e);
2. Exponential Steps: $|d|_e = |e|_{\tilde{\mu}}$ (the number $\tilde{\mu}$ -steps in e), so $|d|_{\text{vsub}_k} = |e|_{\text{vseq}}$;
3. Normal Form: *if u is vsub_k -normal then \underline{u} is vseq -normal.*

Proof p. 51

Corollary 24 (Linear Termination Equivalence of λ_{vsub_k} and λ_{vseq}). *Let t be a vsub_k -term. There is a vsub_k -normalizing derivation d from t iff there is a vseq -normalizing derivation e from \underline{t} . Moreover, $|d|_{\text{vsub}_k} = |e|_{\text{vseq}}$, $|d|_e = |e|_{\tilde{\mu}}$ and $|d|_m = |e|_{\bar{\lambda}}$.*

Structural Equivalence for λ_{vseq} . The equivalence of λ_{vsub} and λ_{vsub_k} relies on the structural equivalence \equiv of λ_{vsub} , so it is natural to wonder how does \equiv look on λ_{vseq} . The structural equivalence \simeq of λ_{vseq} is defined as the closure by evaluation contexts of the following axiom

$$D \langle \tilde{\mu}x.D' \langle \tilde{\mu}y.c \rangle \rangle \simeq_{\tilde{\mu}\tilde{\mu}} D' \langle \tilde{\mu}y.D \langle \tilde{\mu}x.c \rangle \rangle \quad \text{where } x \notin \text{fv}(D') \text{ and } y \notin \text{fv}(D).$$

As expected, \simeq has, with respect to λ_{vseq} , all the properties of \equiv (see Lemma 5). They are formally stated in Appendix B.3, Prop. 52-53.

5 Conclusions and Future Work

This paper proposes Open CBV as a setting halfway between Closed CBV, the simple framework used to model programming languages such as OCaml, and Strong CBV, the less simple setting underling proof assistants such as Coq. Open CBV is a good balance: its rewriting theory is simple—in particular it is strongly confluent, as the one of Closed CBV—and it can be iterated under abstractions to recover Strong CBV, which is not possible with Closed CBV.

We compared four representative calculi for Open CBV, developed with different motivations, and showed that they share the same qualitative (termination/divergence) and quantitative (number of steps) properties with respect to termination. Therefore, they can be considered as different incarnations of the same immaterial setting, justifying the slogan *Open CBV*.

The qualitative equivalences carry semantical consequences: *the adequacy of relational semantics* for the shuffling calculus proved in [8] actually gives a semantic (and type-theoretical, since the relational model can be seen as a non-idempotent intersection type system) characterization of normalizable terms for Open CBV, *i.e.* it extends to the other three calculi. Similarly, the notion of *potential valuability* for Plotkin's CBV λ -calculus, well-studied in [26,24,29,25,23] and recalled at the end of Sect. 2, becomes a robust notion characterizing the same terms in Open (and Strong) CBV.

Quantitatively, we showed that in three out of four calculi for Open CBV, namely λ_{fire} , λ_{vsub} and λ_{vseq} , evaluation takes exactly the same number of β_f -steps, m -steps and λ -steps, respectively. Since such a number is known to be a reasonable time cost model for λ_{fire} [1], the cost model lifts to λ_{vsub} and λ_{vseq} , showing that the cost model is robust, *i.e.* incarnation-independent. For the shuffling calculus λ_{shuf} we obtain a weaker quantitative relationship that does not allow to transfer the cost model. The β_v^b -steps in λ_{shuf} , indeed, match e -steps in λ_{vsub} , but not m -steps. Unfortunately, the two quantities are not necessarily polynomially related, since there is a family of terms that evaluate in exponentially more m -steps than e -steps (details are left to a longer version). Consequently, λ_{shuf} is an incarnation more apt to semantical investigations rather than complexity analyses.

Future Work. This paper is just the first step towards a new, finer understanding of CBV. We plan to pursue at the least the following research directions:

1. *Equational Theories.* The four incarnations are termination equivalent but their rewriting rules do not induce the same equational theory. In particular, λ_{fire} equates more than the others, and probably too much because its theory is not a congruence, *i.e.* it is not stable by context closure. The goal is to establish the relationships between the theories and understand how to smooth the calculi as to make them both *equational* and *termination* equivalent.
2. *Abstract Machines.* Accattoli and Sacerdoti Coen introduce in [1] *reasonable* abstract machines for Open CBV, that is, implementation schemas whose overhead is proven to be polynomial, and even linear. Such machines are quite complex, especially the linear one. Starting from a fine analysis of the overhead, we are currently working on a simpler approach providing cost equivalent but much simpler abstract machines.
3. *From Open CBV to Strong CBV.* We repeatedly said that Strong CBV can be seen as an iteration of Open CBV under abstractions. This is strictly true for λ_{vsub} , λ_{shuf} , and λ_{vseq} , for which the simulations studied here lift to the strong setting. On the contrary, the definition of a good strong λ_{fire} is a subtle open issue. The natural candidate, indeed, is not confluent (but

enjoys uniqueness of normal forms) and normalizes more terms than the other calculi for Strong CBV. Another delicate point is the design and the analysis of abstract machines for Strong CBV, of which there are no examples in the literature (both Grégoire and Leroy’s [13] and Accattoli and Sacerdoti Coen’s [1] study machines for Open CBV only).

4. *Open Bisimulations*. In [19] Lassen studies open (or normal form) bisimulations for CBV. He points out that his bisimilarity is not fully abstract with respect to contextual equivalence, and his counterexamples are all based on stuck β -redexes in Naïve Open CBV. An interesting research direction is to recast his study in Open CBV and see whether full abstraction holds or not.

Acknowledgment This work has been partially supported by the A*MIDEX project ANR-11-IDEX-0001-02 funded by the “Investissements d’Avenir” French Government program, managed by the French National Research Agency (ANR).

References

1. Accattoli, B., Sacerdoti Coen, C.: On the Relative Usefulness of Fireballs. In: LICS 2015. pp. 141–155 (2015)
2. Accattoli, B.: Proof nets and the call-by-value λ -calculus. Theor. Comput. Sci. 606, 2–24 (2015)
3. Accattoli, B., Paolini, L.: Call-by-Value Solvability, revisited. In: FLOPS. pp. 4–16 (2012)
4. Accattoli, B., Sacerdoti Coen, C.: On the Value of Variables. In: WoLLIC 2014. pp. 36–50 (2014)
5. Ariola, Z.M., Bohannon, A., Sabry, A.: Sequent calculi and abstract machines. ACM Trans. Program. Lang. Syst. 31(4) (2009)
6. Barendregt, H.P.: The Lambda Calculus – Its Syntax and Semantics, vol. 103. North-Holland (1984)
7. Blelloch, G.E., Greiner, J.: Parallelism in Sequential Functional Languages. In: FPCA. pp. 226–237 (1995)
8. Carraro, A., Guerrieri, G.: A Semantical and Operational Account of Call-by-Value Solvability. In: FOSSACS 2014. pp. 103–118 (2014)
9. Curien, P.L., Herbelin, H.: The duality of computation. In: ICFP. pp. 233–243 (2000)
10. Curien, P., Munch-Maccagnoni, G.: The Duality of Computation under Focus. In: 6th IFIP, TCS 2010. Proceedings. vol. 323, pp. 165–181. Springer (2010)
11. Dal Lago, U., Martini, S.: The weak lambda calculus as a reasonable machine. Theor. Comput. Sci. 398(1-3), 32–50 (2008)
12. Dyckhoff, R., Lengrand, S.: Call-by-Value lambda-calculus and LJQ. J. Log. Comput. 17(6), 1109–1134 (2007)
13. Grégoire, B., Leroy, X.: A compiled implementation of strong reduction. In: (ICFP ’02). pp. 235–246 (2002)
14. Guerrieri, G.: Head reduction and normalization in a call-by-value lambda-calculus. In: WPTE 2015. pp. 3–17 (2015)
15. Guerrieri, G., Paolini, L., Ronchi Della Rocca, S.: Standardization of a Call-By-Value Lambda-Calculus. In: TLCA 2015. pp. 211–225 (2015)

16. Herbelin, H.: C'est maintenant qu'on calcule: au cœur de la dualité (2005), Habilitation thesis
17. Herbelin, H., Zimmermann, S.: An operational account of Call-by-Value Minimal and Classical λ -calculus in Natural Deduction form. In: TLCA. pp. 142–156 (2009)
18. Jones, N.D., Gomard, C.K., Sestoft, P.: Partial Evaluation and Automatic Program Generation. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1993)
19. Lassen, S.: Eager Normal Form Bisimulation. In: LICS 2005. pp. 345–354 (2005)
20. Lévy, J.J.: Réductions correctes et optimales dans le lambda-calcul. Thèse d'Etat, Univ. Paris VII, France (1978)
21. Maraist, J., Odersky, M., Turner, D.N., Wadler, P.: Call-by-name, Call-by-value, Call-by-need and the Linear λ -Calculus. TCS 228(1-2), 175–210 (1999)
22. Moggi, E.: Computational λ -Calculus and Monads. In: LICS '89. pp. 14–23 (1989)
23. Paolini, L., Pimentel, E., Ronchi Della Rocca, S.: Strong Normalization from an unusual point of view. Theor. Comp. Science 412(20), 1903–1915 (2011)
24. Paolini, L.: Call-by-Value Separability and Computability. In: ICTCS. pp. 74–89 (2002)
25. Paolini, L., Pimentel, E., Ronchi Della Rocca, S.: Lazy strong normalization. In: ITRS '04. Electronic Notes in Theoretical Computer Science, vol. 136C, pp. 103–116 (2005)
26. Paolini, L., Ronchi Della Rocca, S.: Call-by-value Solvability. ITA 33(6), 507–534 (1999)
27. Plotkin, G.D.: Call-by-Name, Call-by-Value and the lambda-Calculus. Theor. Comput. Sci. 1(2), 125–159 (1975)
28. Regnier, L.: Une équivalence sur les lambda-termes. TCS 2(126), 281–292 (1994)
29. Ronchi Della Rocca, S., Paolini, L.: The Parametric λ -Calculus. Springer Berlin Heidelberg (2004)
30. Sabry, A., Felleisen, M.: Reasoning about Programs in Continuation-Passing Style. Lisp and Symbolic Computation 6(3-4), 289–360 (1993)
31. Sabry, A., Wadler, P.: A Reflection on Call-by-Value. ACM Trans. Program. Lang. Syst. 19(6), 916–941 (1997)
32. Sands, D., Gustavsson, J., Moran, A.: Lambda Calculi and Linear Speedups. In: The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones. pp. 60–84 (2002)

Technical Appendix

A Rewriting Theory: Definitions, Notations, and Basic Results

Given a binary relation \rightarrow_r on a set I , the reflexive-transitive (resp. reflexive; transitive; reflexive-transitive and symmetric) closure of \rightarrow_r is denoted by \rightarrow^* (resp. \rightarrow_r^- ; \rightarrow_r^+ ; \simeq_r). The transpose of \rightarrow_r is denoted by \leftarrow_r . A (r -)derivation d from t to u , denoted by $d: t \rightarrow_r^* u$, is a finite sequence $(t_i)_{0 \leq i \leq n}$ of elements of I (with $n \in \mathbb{N}$) s.t. $t = t_0$, $u = t_n$ and $t_i \rightarrow_r t_{i+1}$ for all $1 \leq i < n$;

The number of r -steps of a derivation d , i.e. its length, is denoted by $|d|_r := n$, or simply $|d|$. If $\rightarrow_r = \rightarrow_1 \cup \rightarrow_2$ with $\rightarrow_1 \cap \rightarrow_2 = \emptyset$, $|d|_i$ is the number of \rightarrow_i -steps in d , for $i = 1, 2$. We say that:

- $t \in I$ is r -normal or a r -normal form if $t \not\rightarrow_r u$ for all $u \in I$; $u \in I$ is a r -normal form of t if u is r -normal and $t \rightarrow_r^* u$;
- $t \in I$ is r -normalizable if there is a r -normal $u \in I$ s.t. $t \rightarrow_r^* u$; t is strongly r -normalizable if there is no infinite sequence $(t_i)_{i \in \mathbb{N}}$ s.t. $t_0 = t$ and $t_i \rightarrow_r t_{i+1}$;
- a r -derivation $d: t \rightarrow_r^* u$ is (r -)normalizing if u is r -normal;
- \rightarrow_r is strongly normalizing if all $t \in I$ is strongly r -normalizable;
- \rightarrow_r is strongly confluent if, for all $t, u, s \in I$ s.t. $s \leftarrow_r t \rightarrow_r u$ and $u \neq s$, there is $r \in I$ s.t. $s \rightarrow_r r \leftarrow_r u$; \rightarrow_r is confluent if \rightarrow_r^* is strongly confluent.

Let $\rightarrow_1, \rightarrow_2 \subseteq I \times I$. Composition of relations is denoted by juxtaposition: for instance, $t \rightarrow_1 \rightarrow_2 u$ means that there is $s \in I$ s.t. $t \rightarrow_1 s \rightarrow_2 u$; for any $n \in \mathbb{N}$, $t \rightarrow_1^n u$ means that there is a \rightarrow_1 -derivation with length n ($t = u$ for $n = 0$). We say that \rightarrow_1 and \rightarrow_2 strongly commute if, for any $t, u, s \in I$ s.t. $u \leftarrow_1 t \rightarrow_2 s$, one has $u \neq s$ and there is $r \in I$ s.t. $u \rightarrow_2 r \leftarrow_1 s$. Note that if \rightarrow_1 and \rightarrow_2 strongly commute and $\rightarrow = \rightarrow_1 \cup \rightarrow_2$, then for any derivation $d: t \rightarrow^* u$ the sizes $|d|_1$ and $|d|_2$ are uniquely determined.

The following proposition collects some basic and well-known results of rewriting theory.

Proposition 25. *Let \rightarrow_r be a binary relation on a set I .*

1. *If \rightarrow_r is confluent then:*
 - (a) *every r -normalizable term has a unique r -normal form;*
 - (b) *for all $t, u \in I$, $t \simeq_r u$ iff there is $s \in I$ s.t. $t \rightarrow_r^* s \leftarrow_r^* u$.*
2. *If \rightarrow_r is strongly confluent then \rightarrow_r is confluent and, for any $t \in I$, one has:*
 - (a) *all normalizing r -derivations from t have the same length;*
 - (b) *t is strongly r -normalizable if and only if t is r -normalizable.*

As all incarnations of Open CBV we consider are confluent, the use of Prop. 25.1 is left implicit.

For λ_{fire} and λ_{vsub} , we use Prop. 25.2 and the following more informative version of Hindley–Rosen Lemma, whose proof is just a more accurate reading of the proof in [6, Prop. 3.3.5.(i)]:

Lemma 26 (Strong Hindley–Rosen). *Let $\rightarrow = \rightarrow_1 \cup \rightarrow_2$ be a binary relation on a set I s.t. \rightarrow_1 and \rightarrow_2 are strongly confluent. If \rightarrow_1 and \rightarrow_2 strongly commute, then \rightarrow is strongly confluent and, for any $t \in I$ and any normalizing derivations d and e from t , one has $|d| = |e|$, $|d|_1 = |e|_1$ and $|d|_2 = |e|_2$.*

B Omitted Proofs

B.1 Proofs of Section 2 (Incarnations of Open Call-by-Value)

Naïve Open CBV: Plotkin’s Calculus λ_{Plot}

Remark 27. Since \rightarrow_{β_v} does not reduce under λ ’s, any value is β_v -normal, and so β_y -normal and β_λ -normal, as $\rightarrow_{\beta_y}, \rightarrow_{\beta_\lambda} \subseteq \rightarrow_{\beta_v}$.

Proposition 1. $\rightarrow_{\beta_y}, \rightarrow_{\beta_\lambda}$ and \rightarrow_{β_v} are strongly confluent.

See p. 5

Proof. We prove that \rightarrow_{β_v} is strongly confluent. The proofs that \rightarrow_{β_y} and $\rightarrow_{\beta_\lambda}$ are strongly confluent are perfectly analogous.

So, we prove, by induction on t , that if $t \rightarrow_{\beta_v} u$ and $t \rightarrow_{\beta_v} s$ with $u \neq s$, then there exists t' such that $u \rightarrow_{\beta_v} t'$ and $s \rightarrow_{\beta_v} t'$.

Observe that neither $t \rightarrow_{\beta_v} u$ nor $t \rightarrow_{\beta_v} s$ can be a step at the root: indeed, if $t := (\lambda x.r)v \rightarrow_{\beta_v} r\{x \leftarrow v\} =: u$ and $t \rightarrow_{\beta_v} s$ (or if $t := (\lambda x.r)v \rightarrow_{\beta_v} r\{x \leftarrow v\} =: s$ and $t \rightarrow_{\beta_v} u$), then $u = s$ since $\lambda x.r$ and v are β_v -normal by Remark 27; but this contradicts the hypothesis $u \neq s$. So, according to the definition of $t \rightarrow_{\beta_v} u$ and $t \rightarrow_{\beta_v} s$, there are only four cases.

- *Application Left* for $t \rightarrow_{\beta_v} u$ and $t \rightarrow_{\beta_v} s$, i.e. $t = rq \rightarrow_{\beta_v} pq = u$ and $t = rq \rightarrow_{\beta_v} mq = s$ with $r \rightarrow_{\beta_v} p$ and $r \rightarrow_{\beta_v} m$. By the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists r' such that $p \rightarrow_{\beta_v} r'$ and $m \rightarrow_{\beta_v} r'$. So, setting $t' = r'q$, one has $u = pq \rightarrow_{\beta_v} t'$ and $s = mq \rightarrow_{\beta_v} t'$.
- *Application Right* for $t \rightarrow_{\beta_v} u$ and $t \rightarrow_{\beta_v} s$, i.e. $t = rq \rightarrow_{\beta_v} rp = u$ and $t = rq \rightarrow_{\beta_v} rm = s$ with $q \rightarrow_{\beta_v} p$ and $q \rightarrow_{\beta_v} m$. From the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists q' such that $p \rightarrow_{\beta_v} q'$ and $m \rightarrow_{\beta_v} q'$. So, setting $t' = rq'$, one has $u = rp \rightarrow_{\beta_v} t'$ and $s = rm \rightarrow_{\beta_v} t'$.
- *Application Left* for $t \rightarrow_{\beta_v} u$ and *Application Right* for $t \rightarrow_{\beta_v} s$, i.e. $t = rq \rightarrow_{\beta_v} pq = u$ and $t = rq \rightarrow_{\beta_v} rm = s$ with $r \rightarrow_{\beta_v} p$ and $q \rightarrow_{\beta_v} m$. So, setting $t' = pm$, one has $u = pq \rightarrow_{\beta_v} t'$ and $s = rm \rightarrow_{\beta_v} t'$.
- *Application Right* for $t \rightarrow_{\beta_v} u$ and *Application Left* for $t \rightarrow_{\beta_v} s$, i.e. $t = rq \rightarrow_{\beta_v} rp = u$ and $t = rq \rightarrow_{\beta_v} mq = s$ with $q \rightarrow_{\beta_v} p$ and $r \rightarrow_{\beta_v} m$. So, setting $t' = mp$, one has $u = rp \rightarrow_{\beta_v} t'$ and $s = mq \rightarrow_{\beta_v} t'$. \square

Proofs of Subsection 2.1 (Open CBV 1: the Fireball Calculus λ_{fire})

Remark 28. Inert terms can be equivalently defined as $i ::= x \mid if$ —such a definition is used in the proofs here.

Inert terms that are not variables are referred to as *compound inert terms*.

Lemma 29 (Values and inert terms are β_f -normal).

1. Every value is β_f -normal.
2. Every inert term is β_f -normal.

Proof.

1. Immediate, since \rightarrow_{β_f} does not reduce under λ 's.
2. By induction on the definition of inert term i .
 - If $i = x$ then i is obviously β_f -normal.
 - If $i = i' \lambda x.t$ then i' and $\lambda x.t$ are β_f -normal by *i.h.* and Lemma 29.1 respectively, besides i' is not an abstraction, so i is β_f -normal.
 - Finally, if $i = i' i''$ then i' and i'' are β_f -normal by *i.h.*, moreover i' is not an abstraction, hence i is β_f -normal. \square

See p. 6

Proposition 2 (Open Harmony). *Let $t \in \Lambda$: t is β_f -normal iff t is a fireball.*

Proof.

- \Rightarrow : Proof by induction on $t \in \Lambda$. If t is a value then t is a fireball.
 Otherwise $t = us$ for some terms u and s . Since t is β_f -normal, then u and s are β_f -normal, and either u is not an abstraction or s is not a fireball. By induction hypothesis, u and s are fireballs. Summing up, u is either a variable or an inert term, and s is a fireball, therefore $t = us$ is an inert term and hence a fireball.
- \Leftarrow : By hypothesis, t is either a value or an inert term. If t is a value, then it is β_f -normal by Lemma 29.1. Otherwise t is an inert term and then it is β_f -normal by Lemma 29.2. \square

Lemma 30. *For every $t, t' \in \Lambda$, if $t \rightarrow_{\beta_i} t'$ then $t \neq t'$.*

Proof. By induction on $t \in \Lambda$. According to the definition of $t \rightarrow_{\beta_i} t'$, there are three cases.

- *Step at the root*, i.e. $t = (\lambda x.u)i \rightarrow_{\beta_i} u\{x \leftarrow i\} = t'$: then, since i is not an abstraction, necessarily $t = (\lambda x.u)i \neq u\{x \leftarrow i\} = t'$.
- *Application Left*, i.e. $t = us \rightarrow_{\beta_i} u's = t'$ with $u \rightarrow_{\beta_i} u'$: by *i.h.*, $u \neq u'$ and hence $t = us \neq u's = t'$.
- *Application Right*, i.e. $t = us \rightarrow_{\beta_i} us' = t'$ with $s \rightarrow_{\beta_i} s'$: by *i.h.*, $s \neq s'$ and hence $t = us \neq us' = t'$. \square

See p. 7

Proposition 3 (Basic Properties of λ_{fire}).

1. \rightarrow_{β_i} is strongly normalizing and strongly confluent.
2. $\rightarrow_{\beta_\lambda}$ and \rightarrow_{β_i} strongly commute.
3. \rightarrow_{β_f} is strongly confluent, and all β_f -normalizing derivations d from $t \in \Lambda$ (if any) have the same length $|d|_{\beta_f}$, the same number $|d|_{\beta_\lambda}$ of β_λ -steps, and the same number $|d|_{\beta_i}$ of β_i -steps.

Proof.

1. Strong normalization of \rightarrow_{β_i} follows from general termination properties in the ordinary (i.e. pure, strong, and call-by-name) λ -calculus, as we now explain. Since β_i -steps do not substitute abstractions, they can only cause creations of type 1, according to Lévy's classification of creations of β -redexes [20]. Then β_i -derivations can be seen as special cases of *m-developments* (see Accattoli, B., Kesner, D., *The Permutative λ -Calculus*. In: LPAR. pp. 23-36, 2012), in turn a special case of more famous *superdevelopments*, i.e. reduction sequences reducing only (residuals of) redexes in the original term plus creations of type 1 (*m-developments*) or type 1 and 2 (*superdevelopments*). Both *m-developments* and *superdevelopments* always terminate. Therefore, \rightarrow_{β_i} is strongly normalizing.

Now, we prove that \rightarrow_{β_i} is strongly confluent, that is if $t \rightarrow_{\beta_i} u$ and $t \rightarrow_{\beta_i} s$ with $u \neq s$, then there exists $t' \in \Lambda$ such that $u \rightarrow_{\beta_i} t'$ and $s \rightarrow_{\beta_i} t'$. The proof is by induction on $t \in \Lambda$.

Observe that neither $t \rightarrow_{\beta_i} u$ nor $t \rightarrow_{\beta_i} s$ can be a step at the root: indeed, if $t := (\lambda x.r)i \mapsto_{\beta_i} r\{x \leftarrow i\} := u$ and $t \rightarrow_{\beta_i} s$ (or if $t := (\lambda x.r)i \mapsto_{\beta_i} r\{x \leftarrow i\} := s$ and $t \rightarrow_{\beta_i} u$), then $u = s$ since $\lambda x.r$ and i are β_i -normal by Lemmas 29.1-2 (as $\rightarrow_{\beta_i} \subseteq \rightarrow_{\beta_f}$); but this contradicts the hypothesis $u \neq s$. So, according to the definition of $t \rightarrow_{\beta_i} u$ and $t \rightarrow_{\beta_i} s$, there are only four cases.

- *Application Left* for $t \rightarrow_{\beta_i} u$ and $t \rightarrow_{\beta_i} s$, i.e. $t = rq \rightarrow_{\beta_i} pq = u$ and $t = rq \rightarrow_{\beta_i} mq = s$ with $r \rightarrow_{\beta_i} p$ and $r \rightarrow_{\beta_i} m$. By the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists r' such that $p \rightarrow_{\beta_i} r'$ and $m \rightarrow_{\beta_i} r'$. So, setting $t' = r'q$, one has $u = pq \rightarrow_{\beta_i} t'$ and $s = mq \rightarrow_{\beta_i} t'$.
 - *Application Right* for $t \rightarrow_{\beta_i} u$ and $t \rightarrow_{\beta_i} s$, i.e. $t = rq \rightarrow_{\beta_i} rp = u$ and $t = rq \rightarrow_{\beta_i} rm = s$ with $q \rightarrow_{\beta_i} p$ and $q \rightarrow_{\beta_i} m$. By the hypothesis $u \neq s$ it follows that $p \neq m$. By *i.h.*, there exists q' such that $p \rightarrow_{\beta_i} q'$ and $m \rightarrow_{\beta_i} q'$. So, setting $t' = rq'$, one has $u = rp \rightarrow_{\beta_i} t'$ and $s = rm \rightarrow_{\beta_i} t'$.
 - *Application Left* for $t \rightarrow_{\beta_i} u$ and *Application Right* for $t \rightarrow_{\beta_i} s$, i.e. $t = rq \rightarrow_{\beta_i} pq = u$ and $t = rq \rightarrow_{\beta_i} rm = s$ with $r \rightarrow_{\beta_i} p$ and $q \rightarrow_{\beta_i} m$. So, setting $t' = pm$, one has $u = pq \rightarrow_{\beta_i} t'$ and $s = rm \rightarrow_{\beta_i} t'$.
 - *Application Right* for $t \rightarrow_{\beta_i} u$ and *Application Left* for $t \rightarrow_{\beta_i} s$, i.e. $t = rq \rightarrow_{\beta_i} rp = u$ and $t = rq \rightarrow_{\beta_i} mq = s$ with $q \rightarrow_{\beta_i} p$ and $r \rightarrow_{\beta_i} m$. So, setting $t' = mp$, one has $u = rp \rightarrow_{\beta_i} t'$ and $s = mq \rightarrow_{\beta_i} t'$.
2. We prove, by induction on $t \in \Lambda$, that if $t \rightarrow_{\beta_\lambda} u$ and $t \rightarrow_{\beta_i} s$, then $u \neq s$ and there is $t' \in \Lambda$ such that $u \rightarrow_{\beta_i} t'$ and $s \rightarrow_{\beta_\lambda} t'$.

Observe that neither $t \rightarrow_{\beta_\lambda} u$ nor $t \rightarrow_{\beta_i} s$ can be a step at the root: indeed, if $t := (\lambda x.r)\lambda y.q \mapsto_{\beta_\lambda} r\{x \leftarrow \lambda y.q\} =: u$ (resp. $t := (\lambda x.r)i \mapsto_{\beta_i} r\{x \leftarrow i\} =: s$) then $\lambda y.q$ is not a inert term (resp. i is not an abstraction), moreover $\lambda x.r$ and $\lambda y.q$ (resp. i) are β_i -normal (resp. β_λ -normal) by Prop. 2, as $\rightarrow_{\beta_i} \subseteq \rightarrow_{\beta_f}$ (resp. $\rightarrow_{\beta_\lambda} \subseteq \rightarrow_{\beta_f}$); therefore, t is β_i -normal (resp. β_λ -normal) but this contradicts the hypothesis $t \rightarrow_{\beta_i} s$ (resp. $t \rightarrow_{\beta_\lambda} u$). So, according to the definitions of $t \rightarrow_{\beta_\lambda} u$ and $t \rightarrow_{\beta_i} s$, there are only four cases.

- *Application Left* for both $t \rightarrow_{\beta_\lambda} u$ and $t \rightarrow_{\beta_i} s$, i.e. $t := rq \rightarrow_{\beta_\lambda} pq =: u$ and $t := rq \rightarrow_{\beta_i} mq =: s$ with $r \rightarrow_{\beta_\lambda} p$ and $r \rightarrow_{\beta_i} m$. By *i.h.*, $p \neq m$ and there exists r' such that $p \rightarrow_{\beta_i} r'$ and $m \rightarrow_{\beta_\lambda} r'$. So, $u \neq s$ and, setting $t' := r'q$, one has $u = pq \rightarrow_{\beta_i} t'$ $\beta_\lambda \leftarrow$ $mq = s$.

- *Application Right* for both $t \rightarrow_{\beta_\lambda} u$ and $t \rightarrow_{\beta_i} s$, i.e. $t := rq \rightarrow_{\beta_\lambda} rp =: u$ and $t := rq \rightarrow_{\beta_i} rm =: s$ with $q \rightarrow_{\beta_\lambda} p$ and $q \rightarrow_{\beta_i} m$. By i.h., $p \neq m$ and there exists q' such that $p \rightarrow_{\beta_i} q'$ and $m \rightarrow_{\beta_\lambda} q'$. So, $u \neq s$ and, setting $t' := rq'$, one has $u = rp \rightarrow_{\beta_i} t' \rightarrow_{\beta_\lambda} rm = s$.
 - *Application Left* for $t \rightarrow_{\beta_\lambda} u$ and *Application Right* for $t \rightarrow_{\beta_i} s$, i.e. $t := rq \rightarrow_{\beta_\lambda} pq = u$ and $t := rq \rightarrow_{\beta_i} rm =: s$ with $r \rightarrow_{\beta_\lambda} p$ and $q \rightarrow_{\beta_i} m$. By Lemma 30, $q \neq m$ and hence $u = pq \neq rm = s$. Setting $t' := pm$, one has $u = pq \rightarrow_{\beta_i} t' \rightarrow_{\beta_\lambda} rm = s$.
 - *Application Right* for $t \rightarrow_{\beta_\lambda} u$ and *Application Left* for $t \rightarrow_{\beta_i} s$, i.e. $t := rq \rightarrow_{\beta_\lambda} rp =: u$ and $t := rq \rightarrow_{\beta_i} mq = s$ with $q \rightarrow_{\beta_\lambda} p$ and $r \rightarrow_{\beta_i} m$. By Lemma 30, $r \neq m$ and hence $u = rp \neq mq = s$. Setting $t' := mp$, one has $u = rp \rightarrow_{\beta_i} t' \rightarrow_{\beta_\lambda} mq = s$.
3. It follows immediately from strong confluence of $\rightarrow_{\beta_\lambda}$ (Prop. 1.1) and \rightarrow_{β_i} (Prop. 3.1), the strong commutation of $\rightarrow_{\beta_\lambda}$ and \rightarrow_{β_i} (Prop. 3.2), and Hindley-Rosen (Lemma 26). \square

Proofs of Subsection 2.2 (Open CBV 2: the Value Substitution Calculus λ_{vsub}) Note that vsub -values are vsub -normal (since $\rightarrow_{\text{vsub}}$ does not reduce under λ 's) and closed under substitution (i.e. $v\{x \leftarrow v'\}$ is a vsub -value, for any vsub -values v and v').

See p. 8

Proposition 4 (Basic Properties of λ_{vsub} , [3]).

1. \rightarrow_{m} and \rightarrow_{e} are strongly normalizing (separately).
2. \rightarrow_{m} and \rightarrow_{e} are strongly confluent (separately).
3. \rightarrow_{m} and \rightarrow_{e} strongly commute.
4. $\rightarrow_{\text{vsub}}$ is strongly confluent, and all vsub -normalizing derivations d from $t \in A_{\text{vsub}}$ (if any) have the same length $|d|_{\text{vsub}}$, the same number $|d|_{\text{e}}$ of e -steps, and the same number $|d|_{\text{m}}$ of m -steps.
5. Let $t \in A$. For any vsub -derivation d from t , $|d|_{\text{e}} \leq |d|_{\text{m}}$.

Proof. The statements of Prop. 4 are a refinement of some results proved in [3], where $\rightarrow_{\text{vsub}}$ is denoted by \rightarrow_{w} .

1. In [3, Lemma 3] it has been proved that \rightarrow_{dB} and \rightarrow_{vs} are strongly normalizing, separately. Since $\rightarrow_{\text{m}} \subseteq \rightarrow_{\text{dB}}$ and $\rightarrow_{\text{e}} \subseteq \rightarrow_{\text{vs}}$ (\rightarrow_{dB} and \rightarrow_{vs} are just the extensions of \rightarrow_{m} and \rightarrow_{e} , respectively, obtained by allowing reductions under λ 's), one has that \rightarrow_{m} and \rightarrow_{e} are strongly normalizing, separately.
2. We prove that \rightarrow_{m} is strongly confluent, i.e. if $u \rightarrow_{\text{m}} t \rightarrow_{\text{m}} s$ with $u \neq s$ then there exists $t' \in A_{\text{vsub}}$ such that $u \rightarrow_{\text{m}} t' \rightarrow_{\text{m}} s$. The proof is by induction on the definition of \rightarrow_{m} . Since there $t \rightarrow_{\text{m}} s \neq u$ and the reduction \rightarrow_{m} is weak, there are only eight cases:
 - *Step at the Root* for $t \rightarrow_{\text{m}} u$ and *Application Right* for $t \rightarrow_{\text{m}} s$, i.e. $t := L\langle \lambda x.q \rangle r \mapsto_{\text{m}} L\langle q[x \leftarrow r] \rangle =: u$ and $t \mapsto_{\text{m}} L\langle \lambda x.q \rangle r' =: s$ with $r \rightarrow_{\text{m}} r'$: then, $u \rightarrow_{\text{m}} L\langle q[x \leftarrow r'] \rangle_{\text{m}} \leftarrow s$;

- *Step at the Root for $t \rightarrow_m u$ and Application Left for $t \rightarrow_m s$, i.e.*, for some $n > 0$, $t := (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]r \mapsto_m q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: u$ whereas $t \rightarrow_m (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]r =: s$ with $t_j \rightarrow_m t'_j$ for some $1 \leq j \leq n$: then,

$$u \rightarrow_m q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] \leftarrow_m s;$$

- *Application Left for $t \rightarrow_m u$ and Application Right for $t \rightarrow_m s$, i.e.* $t := rq \rightarrow_m r'q =: u$ and $t \rightarrow_m rq' =: s$ with $r \rightarrow_m r'$ and $q \rightarrow_m q'$: then, $u \rightarrow_m r'q' \leftarrow_m s$;
- *Application Left for both $t \rightarrow_m u$ and $t \rightarrow_m s$, i.e.* $t := rq \rightarrow_m r'q =: u$ and $t \rightarrow_m r''q =: s$ with $r' \leftarrow_m r \rightarrow_m r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_m r_0 \leftarrow_m r''$, hence $u \rightarrow_m r_0q \leftarrow_m s$;
- *Application Right for both $t \rightarrow_m u$ and $t \rightarrow_m s$, i.e.* $t := qr \rightarrow_m qr' =: u$ and $t \rightarrow_m qr'' =: s$ with $r' \leftarrow_m r \rightarrow_m r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_m r_0 \leftarrow_m r''$, hence $u \rightarrow_m qr_0 \leftarrow_m s$;
- *ES Left for $t \rightarrow_m u$ and ES Right for $t \rightarrow_m s$, i.e.* $t := r[x \leftarrow q] \rightarrow_m r'[x \leftarrow q] =: u$ and $t \rightarrow_m r[x \leftarrow q'] =: s$ with $r \rightarrow_m r'$ and $q \rightarrow_m q'$: then, $u \rightarrow_m r'[x \leftarrow q'] \leftarrow_m s$;
- *ES Left for both $t \rightarrow_m u$ and $t \rightarrow_m s$, i.e.* $t := r[x \leftarrow q] \rightarrow_m r'[x \leftarrow q] =: u$ and $t \rightarrow_m r''[x \leftarrow q] =: s$ with $r' \leftarrow_m r \rightarrow_m r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_m r_0 \leftarrow_m r''$, hence $u \rightarrow_m r_0[x \leftarrow q] \leftarrow_m s$;
- *ES Right for both $t \rightarrow_m u$ and $t \rightarrow_m s$, i.e.* $t := q[x \leftarrow r] \rightarrow_m q[x \leftarrow r'] =: u$ and $t \rightarrow_m q[x \leftarrow r''] =: s$ with $r' \leftarrow_m r \rightarrow_m r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_m r_0 \leftarrow_m r''$, hence $u \rightarrow_m q[x \leftarrow r_0] \leftarrow_m s$.

We prove that \rightarrow_e is strongly confluent, i.e. if $u \leftarrow_e t \rightarrow_e s$ with $u \neq s$ then there exists $r \in \Lambda_{\text{vsub}}$ such that $u \rightarrow_e t' \leftarrow_e s$. The proof is by induction on the definition of \rightarrow_e . Since there $t \rightarrow_e s \neq u$ and the reduction \rightarrow_e is weak, there are only eight cases:

- *Step at the Root for $t \rightarrow_e u$ and ES Left for $t \rightarrow_e s$, i.e.* $t := r[x \leftarrow L\langle v \rangle] \mapsto_e L\langle r\{x \leftarrow v\} \rangle =: u$ and $t \mapsto_e r'[x \leftarrow L\langle v \rangle] =: s$ with $r \rightarrow_e r'$: then, $u \rightarrow_e L\langle r'[x \leftarrow v] \rangle \leftarrow_e s$;
- *Step at the Root for $t \rightarrow_e u$ and ES Right for $t \rightarrow_e s$, i.e.*, for some $n > 0$, $t := r[x \leftarrow v[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \mapsto_e r\{x \leftarrow v\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: u$ whereas $t \rightarrow_e r[x \leftarrow v[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] =: s$ with $t_j \rightarrow_e t'_j$ for some $1 \leq j \leq n$: then,

$$u \rightarrow_e r\{x \leftarrow v\}[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] \leftarrow_e s;$$

- *Application Left for $t \rightarrow_e u$ and Application Right for $t \rightarrow_e s$, i.e.* $t := rq \rightarrow_e r'q =: u$ and $t \rightarrow_e rq' =: s$ with $r \rightarrow_e r'$ and $q \rightarrow_e q'$: then, $u \rightarrow_e r'q' \leftarrow_e s$;
- *Application Left for both $t \rightarrow_e u$ and $t \rightarrow_e s$, i.e.* $t := rq \rightarrow_e r'q =: u$ and $t \rightarrow_e r''q =: s$ with $r' \leftarrow_e r \rightarrow_e r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_e r_0 \leftarrow_e r''$, hence $u \rightarrow_e r_0q \leftarrow_e s$;
- *Application Right for both $t \rightarrow_e u$ and $t \rightarrow_e s$, i.e.* $t := qr \rightarrow_e qr' =: u$ and $t \rightarrow_e qr'' =: s$ with $r' \leftarrow_e r \rightarrow_e r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_e r_0 \leftarrow_e r''$, hence $u \rightarrow_e qr_0 \leftarrow_e s$;

- *ES Left* for $t \rightarrow_e u$ and *ES Right* for $t \rightarrow_e s$, i.e. $t := r[x \leftarrow q] \rightarrow_e r'[x \leftarrow q] =: u$ and $t \rightarrow_e r[x \leftarrow q'] =: s$ with $r \rightarrow_e r'$ and $q \rightarrow_e q'$: then, $u \rightarrow_e r'[x \leftarrow q'] \leftarrow_e s$;
- *ES Left* for both $t \rightarrow_e u$ and $t \rightarrow_e s$, i.e. $t := r[x \leftarrow q] \rightarrow_e r'[x \leftarrow q] =: u$ and $t \rightarrow_e r''[x \leftarrow q] =: s$ with $r' \leftarrow_e r \rightarrow_e r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_e r_0 \leftarrow_e r''$, hence $u \rightarrow_e r_0[x \leftarrow q] \leftarrow_e s$;
- *ES Right* for both $t \rightarrow_e u$ and $t \rightarrow_e s$, i.e. $t := q[x \leftarrow r] \rightarrow_e q[x \leftarrow r'] =: u$ and $t \rightarrow_e q[x \leftarrow r''] =: s$ with $r' \leftarrow_e r \rightarrow_e r''$: by i.h., there exists $r_0 \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_e r_0 \leftarrow_e r''$, hence $u \rightarrow_e q[x \leftarrow r_0] \leftarrow_e s$.

Note that in [3, Lemma 11] it has just been proved the strong confluence of $\rightarrow_{\text{vsub}}$, not of \rightarrow_{m} or \rightarrow_e .

3. We show that \rightarrow_e and \rightarrow_{m} strongly commute, i.e. if $u \leftarrow_e t \rightarrow_{\text{m}} s$, then $u \neq s$ and there is $t' \in \Lambda_{\text{vsub}}$ such that $u \rightarrow_{\text{m}} t' \leftarrow_e s$. The proof is by induction on the definition of $t \rightarrow_e u$. The proof that $u \neq s$ is left to the reader. Since the \rightarrow_e and \rightarrow_{m} cannot reduce under λ 's, all **vsub**-values are **m**-normal and **e**-normal. So, there are the following cases.
 - *Step at the Root* for $t \rightarrow_e u$ and *ES Left* for $t \rightarrow_{\text{m}} s$, i.e. $t := r[z \leftarrow L\langle v \rangle] \rightarrow_e L\langle r\{z \leftarrow v\} \rangle =: u$ and $t \rightarrow_{\text{m}} r'[z \leftarrow L\langle v \rangle] =: s$ with $r \rightarrow_{\text{m}} r'$: then $u \rightarrow_{\text{m}} L\langle r'\{z \leftarrow v\} \rangle \leftarrow_e u$;
 - *Step at the Root* for $t \rightarrow_e u$ and *ES Right* for $t \rightarrow_{\text{m}} s$, i.e.

$$\begin{aligned} t &:= r[z \leftarrow v[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \\ &\rightarrow_e r\{z \leftarrow v\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: u \end{aligned}$$

- and $t \rightarrow_{\text{m}} r[z \leftarrow v[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] =: s$ for some $n > 0$, and $t_j \rightarrow_{\text{m}} t'_j$ for some $1 \leq j \leq n$: then, $u \rightarrow_{\text{m}} r\{z \leftarrow v\}[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] \leftarrow_e s$;
- *Application Left* for $t \rightarrow_e u$ and *Application Right* for $t \rightarrow_{\text{m}} s$, i.e. $t := rq \rightarrow_e r'q =: u$ and $t \rightarrow_{\text{m}} rq' =: s$ with $r \rightarrow_e r'$ and $q \rightarrow_{\text{m}} q'$: then, $t \rightarrow_{\text{m}} r'q' \leftarrow_e u$;
- *Application Left* for both $t \rightarrow_e u$ and $t \rightarrow_{\text{m}} s$, i.e. $t := rq \rightarrow_e r'q =: u$ and $t \rightarrow_{\text{m}} r''q =: s$ with $r' \leftarrow_e r \rightarrow_{\text{m}} r''$: by i.h., there exists $p \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_{\text{m}} p \leftarrow_e r''$, hence $u \rightarrow_{\text{m}} pq \leftarrow_e s$;
- *Application Left* for $t \rightarrow_e u$ and *Step at the Root* for $t \rightarrow_{\text{m}} s$, i.e. $t := (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]r \rightarrow_e (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]r =: u$ with $n > 0$ and $t_j \rightarrow_e t'_j$ for some $1 \leq j \leq n$, and $t \rightarrow_{\text{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: s$: then,

$$u \rightarrow_{\text{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] \leftarrow_e s;$$

- *Application Right* for $t \rightarrow_e u$ and *Application Left* for $t \rightarrow_{\text{m}} s$, i.e. $t := qr \rightarrow_e qr' =: u$ and $t \rightarrow_{\text{m}} q'r =: s$ with $r \rightarrow_e r'$ and $q \rightarrow_{\text{m}} q'$: then, $u \rightarrow_{\text{m}} q'r' \leftarrow_e s$;
- *Application Right* for both $t \rightarrow_e u$ and $t \rightarrow_{\text{m}} s$, i.e. $t := qr \rightarrow_e qr' =: u$ and $t \rightarrow_{\text{m}} qr'' =: s$ with $r' \leftarrow_e r \rightarrow_{\text{m}} r''$: by i.h., there exists $p \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_{\text{m}} p \leftarrow_e r''$, hence $u \rightarrow_{\text{m}} qp \leftarrow_e s$;

- *Application Right* for $t \rightarrow_e u$ and *Step at the Root* for $t \rightarrow_m s$, i.e. $t := L\langle\lambda x.q\rangle r \rightarrow_e L\langle\lambda x.q\rangle r' =: u$ with $r \rightarrow_e r'$, and $t \rightarrow_m L\langle q[x \leftarrow r] \rangle =: s$: then, $u \rightarrow_m L\langle q[x \leftarrow r'] \rangle \leftarrow_e s$;
 - *ES Left* for $t \rightarrow_e u$ and *ES Right* for $t \rightarrow_m s$, i.e. $t := r[x \leftarrow q] \rightarrow_e r'[x \leftarrow q] =: u$ and $t \rightarrow_m r[x \leftarrow q'] =: s$ with $r \rightarrow_e r'$ and $q \rightarrow_m q'$: then, $u \rightarrow_m r'[x \leftarrow q'] \leftarrow_e s$;
 - *ES Left* for both $t \rightarrow_e u$ and $t \rightarrow_m s$, i.e. $t := r[x \leftarrow q] \rightarrow_e r'[x \leftarrow q] =: u$ and $t \rightarrow_m r''[x \leftarrow q] =: s$ with $r' \leftarrow_e r \rightarrow_m r''$: by i.h., there exists $p \in \Lambda_{\text{vsub}}$ such that $r' \rightarrow_m p \leftarrow_e r''$, hence $u \rightarrow_m p[x \leftarrow q] \leftarrow_e s$;
 - *ES Right* for $t \rightarrow_e u$ and *ES Left* for $t \rightarrow_m s$, i.e. $t := q[x \leftarrow r] \rightarrow_e q[x \leftarrow r'] =: u$ and $t \rightarrow_m q'[x \leftarrow r] =: s$ with $r \rightarrow_e r'$ and $q \rightarrow_m q'$: then, $u \rightarrow_m q'[x \leftarrow r'] \leftarrow_e s$;
 - *ES Right* for both $t \rightarrow_e u$ and $t \rightarrow_m s$, i.e. $t := q[x \leftarrow r] \rightarrow_e q[x \leftarrow r'] =: u$ and $t \rightarrow_m q[x \leftarrow r''] =: s$ with $r \leftarrow_e r' \rightarrow_m r''$: by i.h., there exists $p \in \Lambda_{\text{vsub}}$ such that $r \rightarrow_m p \leftarrow_e r''$, hence $u \rightarrow_m q[x \leftarrow p] \leftarrow_e s$.
4. It follows immediately from strong confluence of \rightarrow_m and \rightarrow_e (Prop. 4.1), strong commutation of \rightarrow_m and \rightarrow_e (Prop. 4.2) and Hindley-Rosen (Lemma 26). A different proof of the strong confluence of $\rightarrow_{\text{vsub}}$ (without information about the number of steps) is in [3, Lemma 11].
5. The intuition behind the proof is that any m -step creates a new ES, any e -step erases an ES. Formally, let $u \in \Lambda_{\text{vsub}}$ such that $d: t \rightarrow_{\text{vsub}}^* u$. We prove by induction on $|d|_{\text{vsub}} \in \mathbb{N}$ that $|d|_e = |d|_m - |u|_{\text{ES}}$ (where $|u|_{\text{ES}}$ is the number of ES in u) and any vsub -value that is a subterm of u is a value (without ES).
- If $|d|_{\text{vsub}} = 0$, then $u = t \in \Lambda$, then we can conclude.
- Suppose $|d|_{\text{vsub}} > 0$: then, d is the concatenation of $d': t \rightarrow_{\text{vsub}}^* s$ and $s \rightarrow_{\text{vsub}} u$, for some $s \in \Lambda_{\text{vsub}}$. By i.h., $|d'|_e = |d'|_m - |s|_{\text{ES}}$ and that every vsub -value that is a subterm of s is a value (without ES). There are two cases:
- $s := E\langle r[x \leftarrow L\langle v \rangle] \rangle \rightarrow_e E\langle L\langle r\{x \leftarrow v\} \rangle \rangle =: u$, then $|d|_m = |d'|_m$ and $|s|_{\text{ES}} = |u|_{\text{ES}} + 1$, since $|v|_{\text{ES}} = 0$ by i.h.; therefore $|d|_e = |d'|_e + 1 = |d'|_m - |s|_{\text{ES}} + 1 = |d|_m - |u|_{\text{ES}}$ and any vsub -value that is a subterm of u is a value (without ES).
 - $s := E\langle L\langle \lambda x.r \rangle q \rangle \rightarrow_m E\langle L\langle r[x \leftarrow q] \rangle \rangle =: u$, then $|u|_{\text{ES}} = |s|_{\text{ES}} + 1$ and $|d|_m = |d'|_m + 1$, therefore $|d|_e = |d'|_e = |d'|_m - |s|_{\text{ES}} = |d|_m - |u|_{\text{ES}}$. Moreover, the new occurrence of ES $[x \leftarrow q]$ in u cannot be under the scope of a λ , otherwise the redex in s which is fired in the m -step would be under the scope of a λ , but this is impossible since \rightarrow_m is a weak reduction. So, any vsub -value that is a subterm of u is a value (without ES).

□

Proof of Subsection 2.3 (Open CBV 3: the Shuffling Calculus λ_{shuf})

Definition 31 (Occurrences). For all $t \in \Lambda$, let $[t]_\lambda$ be the number of occurrences of λ in t , and $[t]_x$ be the number of free occurrences of the variable x in t , and $\text{sub}_u(t)$ be the number of occurrences in t of the term u .

Remark 32. Since $\rightarrow_{\beta_v^b}$ and \rightarrow_{σ^b} do not reduce under λ 's without argument, every value is β_v^b -normal and σ^b -normal, and hence **shuf**-normal.

Remark 33. The reduction \rightarrow_{σ^b} is just the closure under balanced contexts of the binary relation $\mapsto_{\sigma} = \mapsto_{\sigma_1} \cup \mapsto_{\sigma_3}$ on Λ (see definitions in Fig. 4).

Lemma 34. *Let $t, t' \in \Lambda$.*

1. *For every value v , if $t \rightarrow_{\sigma^b} t'$ then $(\lambda x.t')v \neq t\{x \leftarrow v\}$.*
2. *If $t \rightarrow_{\sigma^b} t'$ then $t \neq t'$.*
3. *For every value v , one has $t\{x \leftarrow v\} \neq \lambda x.tv$.*

Proof.

1. By induction on the definition of $t \rightarrow_{\sigma^b} t'$, using Remark 33.
2. In [8, Proposition 2] it has been proved that there exists a size $\# : \Lambda \rightarrow \mathbb{N}$ such that if $t \rightarrow_{\sigma} t'$ then $\#(t) > \#(t')$, where \rightarrow_{σ} is just the extension of \rightarrow_{σ^b} obtained by allowing reductions under λ 's. Therefore, $\rightarrow_{\sigma^b} \subseteq \rightarrow_{\sigma}$ and hence if $t \rightarrow_{\sigma^b} t'$ then $\#(t) > \#(t')$, in particular $t \neq t'$.
3. According to Definition 31, $[t\{x \leftarrow v\}]_{\lambda} = [t]_{\lambda} + [v]_{\lambda} \cdot [t]_x$ and $[\lambda x.tv]_{\lambda} = 1 + [t]_{\lambda} + [v]_{\lambda}$, and $[t\{x \leftarrow v\}]_x = [t]_x \cdot [v]_x$ and $[\lambda x.tv]_x = 0$. Suppose $t\{x \leftarrow v\} = \lambda x.tv$: then, $[t\{x \leftarrow v\}]_{\lambda} = [\lambda x.tv]_{\lambda}$ and $[t\{x \leftarrow v\}]_x = [\lambda x.tv]_x$, thus

$$[v]_{\lambda} \cdot [t]_x = 1 + [v]_{\lambda} \quad [t]_x \cdot [v]_x = 0. \quad (3)$$

The only solution to the first equation of (3) is $[v]_{\lambda} = 1$ and $[t]_x = 2$, whence $[v]_x = 0$ according to the second equation of (3). As $x \notin \text{fv}(v)$, one has $\text{sub}_v(\lambda x.tv) = 1 + \text{sub}_v(t)$ and $\text{sub}_v(t\{x \leftarrow v\}) = \text{sub}_v(t) + [t]_x = \text{sub}_v(t) + 2$, therefore $\text{sub}_v(\lambda x.tv) \neq \text{sub}_v(t\{x \leftarrow v\})$ and hence $\lambda x.tv \neq t\{x \leftarrow v\}$. Contradiction. \square

See p. 10

Proposition 6 (Basic Properties of λ_{shuf} , [8]).

1. *Let $t, u, s \in \Lambda$: if $t \rightarrow_{\beta_v^b} u$ and $t \rightarrow_{\sigma^b} s$ then $u \neq s$.*
2. *\rightarrow_{σ^b} is strongly normalizing and (not strongly) confluent.*
3. *$\rightarrow_{\text{shuf}}$ is (not strongly) confluent.*
4. *Let $t \in \Lambda$: t is strongly **shuf**-normalizable iff t is **shuf**-normalizable.*

Proof.

1. By induction on $t \in \Lambda$. According to the definition of $t \rightarrow_{\sigma^b} s$ and Remark 33, the following cases are impossible.
 - *Step at the root for $t \rightarrow_{\beta_v^b} u$* and either the *Step at the root* or the *Application Left* or the *Application Right* for $t \rightarrow_{\sigma^b} s$. Indeed, if $t = (\lambda x.r)v \mapsto_{\beta_v} r\{x \leftarrow v\} = u$ then $\lambda x.r$ and v are σ^b -normal by Remark 32; moreover t is neither a σ_1 -redex nor a σ_3 -redex, because $\lambda x.r$ and v , respectively, are not applications.

- *Application Left* for $t \rightarrow_{\beta_v^b} u$ and *Step inside a β -context* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} pq = u$ with $r \rightarrow_{\beta_v^b} p$, and $t = (\lambda x.r')q \rightarrow_{\sigma^b} (\lambda x.m)q = s$ with $r = \lambda x.r'$ and $r' \rightarrow_{\sigma^b} m$. Indeed r is β_v^b -normal by Remark 32.
- *Step inside a β -context* for $t \rightarrow_{\beta_v^b} u$ and *Application Left* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\sigma^b} pq = s$ with $r \rightarrow_{\sigma^b} p$, and $t = (\lambda x.r')q \rightarrow_{\beta_v^b} (\lambda x.m)q = u$ with $r = \lambda x.r'$ and $r' \rightarrow_{\beta_v^b} m$. Indeed r is σ^b -normal by Remark 32.

Therefore, according to the definition of $t \rightarrow_{\sigma^b} s$ and Remark 33, there are “only” eleven cases.

- *Step at the root* for $t \rightarrow_{\beta_v^b} u$ and *Step inside a β -context* for $t \rightarrow_{\sigma^b} s$, i.e. $t = (\lambda x.r)v \mapsto_{\beta_v} r\{x \leftarrow v\} = u$ and $t = (\lambda x.r)v \rightarrow_{\sigma^b} (\lambda x.r')v = s$ with $r \rightarrow_{\sigma^b} r'$. By Lemma 34.1, $u \neq s$.
- *Application Left* for $t \rightarrow_{\beta_v^b} u$ and *Step at the root* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} pq = u$ with $r \rightarrow_{\beta_v^b} p$, and $t \mapsto_{\sigma} s$ (see Remark 33). It is impossible that $t \mapsto_{\sigma_3} s$, otherwise r would be a value and hence β_v^b -normal by Remark 32, but this contradicts that $r \rightarrow_{\beta_v^b} p$. Thus, $t = (\lambda x.r')r''q \mapsto_{\sigma_1} (\lambda x.r'q)r'' = s$ with $x \notin \text{fv}(q)$ and $r = (\lambda x.r')r''$. We claim that $u \neq s$. Indeed, if $u = s$ then $q = r''$ and $p = \lambda x.r'q$ with $r = (\lambda x.r')q \rightarrow_{\beta_v^b} \lambda x.r'q = p$, hence necessarily $r \mapsto_{\beta_v} p$ (i.e. $r \rightarrow_{\beta_v^b} p$ by a step at the root) and thus q is a value and $\lambda x.r'q = p = r'\{x \leftarrow q\}$, but this is impossible by Lemma 34.3.
- *Application Left* for $t \rightarrow_{\beta_v^b} u$ and $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} pq = u$ and $t = rq \rightarrow_{\sigma^b} mq = s$ with $r \rightarrow_{\beta_v^b} p$ and $r \rightarrow_{\sigma^b} m$. By i.h., $p \neq m$ and hence $u = pq \neq mq = s$.
- *Application Left* for $t \rightarrow_{\beta_v^b} u$ and *Application Right* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} pq = u$ and $t = rq \rightarrow_{\sigma^b} rm = s$, with $r \rightarrow_{\beta_v^b} p$ and $q \rightarrow_{\sigma^b} m$. By Lemma 34.2, $q \neq m$ and hence $u = pq \neq rm = s$.
- *Application Right* for $t \rightarrow_{\beta_v^b} u$ and *Step at the root* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} rp = u$ with $q \rightarrow_{\beta_v^b} p$, and $t \mapsto_{\sigma} s$ (see Remark 33).
 - If $t \mapsto_{\sigma_1} s$ then $t = (\lambda x.r')r''q \mapsto_{\sigma_1} (\lambda x.r'q)r'' = s$ with $x \notin \text{fv}(q)$ and $r = (\lambda x.r')r''$. We claim that $u \neq s$. Indeed, if $u = s$ then $p = r''$ and $r = \lambda x.r'q$, therefore $(\lambda x.r')p = r = \lambda x.r'q$ which is impossible.
 - If $t \mapsto_{\sigma_3} s$ then $t = r((\lambda x.q')q'') \mapsto_{\sigma_3} (\lambda x.rq')q'' = s$ where r is a value, $x \notin \text{fv}(r)$ and $q = (\lambda x.q')q''$. We claim that $u \neq s$. Indeed, if $u = s$ then $r = \lambda x.rq'$ which is impossible.
- *Application Right* for $t \rightarrow_{\beta_v^b} u$ and $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} pq = u$ and $t = rq \rightarrow_{\sigma^b} mq = s$ with $q \rightarrow_{\beta_v^b} p$ and $q \rightarrow_{\sigma^b} m$. By i.h., $p \neq m$ and hence $u = rp \neq rm = s$.
- *Application Right* for $t \rightarrow_{\beta_v^b} u$ and *Application Left* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} rp = u$ and $t = rq \rightarrow_{\sigma^b} mq = s$, with $q \rightarrow_{\beta_v^b} p$ and $r \rightarrow_{\sigma^b} m$. By Lemma 34.2, $r \neq m$ and hence $u = rp \neq mq = s$.
- *Application Right* for $t \rightarrow_{\beta_v^b} u$ and *Step inside a β -context* for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\beta_v^b} rp = u$ with $q \rightarrow_{\beta_v^b} p$, and $t = (\lambda x.r')q \rightarrow_{\sigma^b} (\lambda x.m)q = s$ with $r = \lambda x.r'$ and $r' \rightarrow_{\sigma^b} m$. By Lemma 34.2, $r' \neq m$ whence $r = \lambda x.r' \neq \lambda x.m$ and thus $u \neq s$.
- *Step inside a β -context* for $t \rightarrow_{\beta_v^b} u$ and *Step at the root* for $t \rightarrow_{\sigma^b} s$, i.e. $t = (\lambda x.r)q \rightarrow_{\beta_v^b} (\lambda x.r')q = u$ with $r \rightarrow_{\beta_v^b} r'$, and $t \mapsto_{\sigma} s$ (see

- Remark 33). It is impossible that $t = (\lambda x.r)q \mapsto_{\sigma_1} s$ because $\lambda x.r$ is not an application. Thus, $t = (\lambda x.r)((\lambda y.q')q'') \mapsto_{\sigma_3} (\lambda y.(\lambda x.r)q')q'' = s$ with $q = (\lambda y.q')q''$ and $y \notin \text{fv}(\lambda x.r)$, therefore $q \neq q''$ and hence $u \neq s$.
- *Step inside a β -context for $t \rightarrow_{\beta_v} u$ and Application Right for $t \rightarrow_{\sigma^b} s$, i.e. $t = rq \rightarrow_{\sigma^b} rp = s$ with $q \rightarrow_{\sigma^b} p$, and $t = (\lambda x.r')q \rightarrow_{\beta_v} (\lambda x.m)q = u$ with $r = \lambda x.r'$ and $r' \rightarrow_{\beta_v} m$. By Lemma 34.2, $q \neq p$ whence $u \neq s$.*
 - *Step inside a β -context for $t \rightarrow_{\beta_v} u$ and $t \rightarrow_{\sigma^b} s$, i.e. $t = (\lambda x.r)q \rightarrow_{\beta_v} (\lambda x.p)q = u$ and $t = (\lambda x.r)q \rightarrow_{\sigma^b} (\lambda x.m)q = s$ with $r \rightarrow_{\beta_v} p$ and $r \rightarrow_{\sigma^b} m$. By i.h., $p \neq m$ and hence $u \neq s$.*
2. In [8, Proposition 2] it has been proved that \rightarrow_{σ} is strongly normalizing, where \rightarrow_{σ} is just the extension of \rightarrow_{σ^b} obtained by allowing reductions under λ 's. Therefore, $\rightarrow_{\sigma^b} \subseteq \rightarrow_{\sigma}$ and hence \rightarrow_{σ^b} is strongly normalizing. The (not strong) confluence of \rightarrow_{σ^b} has been proved in [8, Lemma 9.ii], where \rightarrow_{σ^b} is denoted by $\rightarrow_{\mathbf{w}[\sigma]}$.
3. See [8, Proposition 10], where $\rightarrow_{\text{shuf}}$ is denoted by $\rightarrow_{\mathbf{w}}$.
4. See [8, Theorem 24], where $\rightarrow_{\text{shuf}}$ is denoted by $\rightarrow_{\mathbf{w}}$. □

Proofs of Subsection 2.4 (Open CBV 4: the Value Sequent Calculus λ_{vseq})

We aim to prove the strong confluence of $\rightarrow_{\text{vseq}}$.

Note that values are closed under substitution: for all values $v, v', v\{x \leftarrow v'\}$ is a value. Moreover, values are $\bar{\lambda}$ -, $\tilde{\mu}$ - and vseq -normal.

Definition 35. For any $r \in \{\bar{\lambda}, \tilde{\mu}, \text{vseq}\}$, given two environments e and e' , we define $e \rightarrow_r e'$ by induction on e . If $e = \epsilon$ then there is no e' such that $e \rightarrow_r e'$. If $e = \tilde{\mu}x.c$ then $e' = \tilde{\mu}x.c'$ and $c \rightarrow_r c'$. If $e = v \cdot e_0$ then $e' = v \cdot e'_0$ and $e_0 \rightarrow_r e'_0$.

Remark 36. Let c and c' be commands and $r \in \{\bar{\lambda}, \tilde{\mu}, \text{vseq}\}$. One has $c \rightarrow_r c'$ iff

- either $c = \langle \lambda x.c_0 \mid v \cdot e \rangle$ and $c' = \langle v \mid (\tilde{\mu}x.c_0) @ e \rangle$,
- or $c = \langle v \mid \tilde{\mu}x.c_0 \rangle$ and $c' = c_0\{x \leftarrow v\}$,
- or $c = \langle v \mid e \rangle$, $c' = \langle v \mid e' \rangle$ and $e \rightarrow_r e'$.

Lemma 37 (Substitution). Let c and c' be commands, e and e' be environments, v be a value and x be a variable. Let $r \in \{\bar{\lambda}, \tilde{\mu}, \text{vseq}\}$.

1. If $e \rightarrow_r e'$ then $e\{x \leftarrow v\} \rightarrow_r e'\{x \leftarrow v\}$;
2. If $c \rightarrow_r c'$ then $c\{x \leftarrow v\} \rightarrow_r c'\{x \leftarrow v\}$.

Proof. Both points are proved simultaneously by mutual induction on c and e . Cases:

1. *Step at the root for $c \rightarrow_{\bar{\lambda}} c'$, i.e. $c := \langle \lambda y.c_0 \mid v' \cdot e_0 \rangle \mapsto_{\bar{\lambda}} \langle v' \mid (\tilde{\mu}y.c_0) @ e_0 \rangle =: c'$.* We can suppose without loss of generality that $y \notin \text{fv}(v) \cup \{x\}$. So, $c\{x \leftarrow v\} = \langle \lambda y.c_0\{x \leftarrow v\} \mid v'\{x \leftarrow v\} \cdot e_0\{x \leftarrow v\} \rangle \rightarrow_{\bar{\lambda}} \langle v'\{x \leftarrow v\} \mid (\tilde{\mu}y.c_0\{x \leftarrow v\}) @ e_0\{x \leftarrow v\} \rangle = c'\{x \leftarrow v\}$.
2. *Step at the root for $c \rightarrow_{\tilde{\mu}} c'$, i.e. $c := \langle v' \mid \tilde{\mu}y.c_0 \rangle \mapsto_{\tilde{\mu}} c_0\{y \leftarrow v'\} =: c'$.* We can suppose without loss of generality that $y \notin \text{fv}(v) \cup \{x\}$. So, $c\{x \leftarrow v\} = \langle v'\{x \leftarrow v\} \mid \tilde{\mu}y.c_0\{x \leftarrow v\} \rangle \rightarrow_{\tilde{\mu}} c_0\{x \leftarrow v\}\{y \leftarrow v'\{x \leftarrow v\}\} = c'\{x \leftarrow v\}$.

3. *Environment step for $c \rightarrow_r c'$, i.e. $c := \langle v' | e \rangle \rightarrow_r \langle v' | e' \rangle =: c'$ with $e \rightarrow_r e'$:* by *i.h.*, $e\{x \leftarrow v\} \rightarrow_r e'\{x \leftarrow v\}$ and hence $c\{x \leftarrow v\} = \langle v'\{x \leftarrow v\} | e\{x \leftarrow v\} \rangle \rightarrow_r \langle v'\{x \leftarrow v\} | e'\{x \leftarrow v\} \rangle = c'\{x \leftarrow v\}$ according to Remark 36.
4. *$\tilde{\mu}$ -environment step for $e \rightarrow_r e'$, i.e. $e := \tilde{\mu}y.c \rightarrow_r \tilde{\mu}y.c' =: e'$ with $c \rightarrow_r c'$.* We can suppose without loss of generality that $y \notin \text{fv}(v) \cup \{x\}$. By *i.h.* $c\{x \leftarrow v\} \rightarrow_r c'\{x \leftarrow v\}$, and thus $e\{x \leftarrow v\} = \tilde{\mu}y.c\{x \leftarrow v\} \rightarrow_r \tilde{\mu}y.c'\{x \leftarrow v\} = e'\{x \leftarrow v\}$ according to Definition 35.
5. *Environment step for $e \rightarrow_r e'$, i.e. $e := v' \cdot e_0 \rightarrow_r v' \cdot e'_0 =: e'$ with $e_0 \rightarrow_r e'_0$:* by *i.h.*, $e_0\{x \leftarrow v\} \rightarrow_r e'_0\{x \leftarrow v\}$ and hence $e\{x \leftarrow v\} = v'\{x \leftarrow v\} \cdot e_0\{x \leftarrow v\} \rightarrow_r v'\{x \leftarrow v\} \cdot e'_0\{x \leftarrow v\} = e'\{x \leftarrow v\}$ according to Definition 35. \square

Lemma 38 (Append). *Let $r \in \{\bar{\lambda}, \tilde{\mu}, \text{vseq}\}$, c be a command and e_0, e and e' be environments. If $e \rightarrow_r e'$ then $e_0 @ e \rightarrow_r e_0 @ e'$ and $c @ e \rightarrow_r c @ e'$.*

Proof. We prove simultaneously that $e_0 @ e \rightarrow_r e_0 @ e'$ and $c @ e \rightarrow_r c @ e'$ by mutual induction on c and e_0 . Cases:

1. $c = \langle v | e_0 \rangle$: by *i.h.*, $e_0 @ e \rightarrow_r e_0 @ e'$. Thus, $c @ e = \langle v | e_0 @ e \rangle \rightarrow_r \langle v | e_0 @ e' \rangle = c @ e'$ according to Remark 36.
2. $e_0 = \epsilon$: then, $e_0 @ e = e \rightarrow_r e' = e_0 @ e'$.
3. $e_0 = v_0 \cdot e'_0$: by *i.h.*, $e'_0 @ e \rightarrow_r e'_0 @ e'$. Hence, $e_0 @ e = v_0 \cdot (e'_0 @ e) \rightarrow_r v_0 \cdot (e'_0 @ e') = e_0 @ e'$ according to Definition 35.
4. $e_0 = \tilde{\mu}y.c$: we can suppose without loss of generality that $y \notin \text{fv}(v) \cup \text{fv}(e) \cup \{x\}$, whence $y \notin \text{fv}(e')$. By *i.h.*, $c @ e \rightarrow_r c @ e'$. Hence, $e_0 @ e = \tilde{\mu}y.c @ e \rightarrow_r \tilde{\mu}y.c @ e' = e_0 @ e'$ according to Definition 35. \square

Lemma 39 (Append Commutes).

1. *Evaluation Contexts: $C\langle c \rangle @ e = C\langle c @ e \rangle$ and $D\langle e' \rangle @ e = D\langle e' @ e \rangle$.*
2. *Rewriting Steps in Commands and Environments: if $c \rightarrow_{\bar{\lambda}\tilde{\mu}} c'$ (resp. $e \rightarrow_{\bar{\lambda}\tilde{\mu}} e'$) then $c @ e_0 \rightarrow_{\text{vseq}} c' @ e_0$ (resp. $e @ e_0 \rightarrow_{\text{vseq}} e' @ e_0$).*

Proof.

1. By mutual induction on C and D (see Fig. 5). Cases:
 - $C = \langle \cdot \rangle$: then, $C\langle c \rangle @ e = c @ e = C\langle c @ e \rangle$.
 - $C = D\langle \tilde{\mu}x.C' \rangle$: we can suppose without loss of generality that $x \notin \text{fv}(e)$. So, $C\langle c \rangle @ e = D\langle \tilde{\mu}x.C'\langle c \rangle \rangle @ e \stackrel{i.h.}{=} D\langle (\tilde{\mu}x.C'\langle c \rangle) @ e \rangle = D\langle \tilde{\mu}x.(C'\langle c \rangle @ e) \rangle \stackrel{i.h.}{=} D\langle \tilde{\mu}x.C'\langle c @ e \rangle \rangle = C\langle c @ e \rangle$, with we have applied the *i.h.* the first time to D , the second time to C' .
 - $D = \langle v | \langle \cdot \rangle \rangle$: then, $D\langle e' \rangle @ e = \langle v | e' \rangle @ e = \langle v | e' @ e \rangle = D\langle e' @ e \rangle$.
 - $D = D'\langle v \cdot \langle \cdot \rangle \rangle$: one has $D\langle e' \rangle @ e = D'\langle v \cdot e' \rangle @ e \stackrel{i.h.}{=} D'\langle (v \cdot e') @ e \rangle = D'\langle v \cdot (e' @ e) \rangle$.
2. By mutual induction on c and e . According to Remark 36, there are three cases for $c \rightarrow_{\text{vseq}} c'$:
 - either $c = \langle \lambda x.c_0 | v \cdot e \rangle \rightarrow_{\text{vseq}} \langle v | (\tilde{\mu}x.c_0) @ e \rangle = c'$: then, $c @ e_0 = \langle \lambda x.c_0 | v \cdot (e @ e_0) \rangle \rightarrow_{\text{vseq}} \langle v | \tilde{\mu}x.c_0 @ (e @ e_0) \rangle = \langle v | \tilde{\mu}x.(c_0 @ e) @ e_0 \rangle = c' @ e_0$, where the next-to-last identity holds by Lemma 39.1 taking $D = \langle v | \tilde{\mu}x.c_0 @ \langle \cdot \rangle \rangle$;

- or $c = \langle v \mid \tilde{\mu}x.c_0 \rangle \rightarrow_{\text{vseq}} c_0\{x \leftarrow v\} = c'$: we can suppose without loss of generality that $x \notin \text{fv}(e_0)$; so, $c@e_0 = \langle v \mid \tilde{\mu}x.(c_0@e_0) \rangle \rightarrow_{\text{vseq}} (c_0@e_0)\{x \leftarrow v\} = c'@e_0$;
- or $c = \langle v \mid e \rangle \rightarrow_{\text{vseq}} \langle v \mid e' \rangle = c'$ with $e \rightarrow_{\text{vseq}} e'$: then, $c@e_0 = \langle v \mid e@e_0 \rangle \rightarrow_{\text{vseq}} \langle v \mid e'@e_0 \rangle = c'@e_0$ by Remark 36.

According to Definition 35, there are only two cases for $e \rightarrow_{\text{vseq}} e'$:

- either $e = \tilde{\mu}x.c$ and $e' = \tilde{\mu}x.c'$ and $c \rightarrow_{\text{vseq}} c'$: we can suppose without loss of generality that $x \notin \text{fv}(e_0)$; by *i.h.*, $c@e_0 \rightarrow_{\text{vseq}} c'@e_0$ and hence $e@e_0 = \tilde{\mu}x.(c@e_0) \rightarrow_{\text{vseq}} \tilde{\mu}x.(c'@e_0) = e'@e_0$;
- or $e = v.e_1$ and $e' = v.e'_1$ with $e_1 \rightarrow_{\text{vseq}} e'_1$; by *i.h.*, $e@e_1 \rightarrow_{\tilde{\lambda}\tilde{\mu}} e'@e_1$ and hence $e@e_0 = v.(e_1@e_0) \rightarrow_{\tilde{\lambda}\tilde{\mu}} v.(e'_1@e_0) = e'@e_0$. \square

See p. 12

Proposition 7 (Basic properties of $\tilde{\lambda}\tilde{\mu}$).

1. $\rightarrow_{\tilde{\lambda}}$ is strongly normalizing and strongly confluent.
2. $\rightarrow_{\tilde{\mu}}$ is strongly normalizing and strongly confluent.
3. $\rightarrow_{\tilde{\lambda}}$ and $\rightarrow_{\tilde{\mu}}$ strongly commute.
4. $\rightarrow_{\text{vseq}}$ is strongly confluent, and all *vseq*-normalizing derivations d from a command c or an environment e (if any) have the same length $|d|_{\text{vseq}}$, the same number $|d|_{\tilde{\mu}}$ of $\tilde{\mu}$ -steps, and the same number $|d|_{\tilde{\lambda}}$ of $\tilde{\lambda}$ -steps.

Proof.

1. Note that if $c \rightarrow_{\tilde{\lambda}} c'$ then the number of occurrences of λ in c' is strictly less than in c : this is enough to prove that $\rightarrow_{\tilde{\lambda}}$ is strongly normalizing. Concerning the strong confluence of $\rightarrow_{\tilde{\lambda}}$, we prove that
 - (a) (commands) if $c \rightarrow_{\tilde{\lambda}} c_1$ and $c \rightarrow_{\tilde{\lambda}} c_2$ with $c_1 \neq c_2$, then there exists c' such that $c_1 \rightarrow_{\tilde{\lambda}} c'$ and $c_2 \rightarrow_{\tilde{\lambda}} c'$;
 - (b) (environments) if $e \rightarrow_{\tilde{\lambda}} e_1$ and $e \rightarrow_{\tilde{\lambda}} e_2$ with $e_1 \neq e_2$, then there exists e' such that $e_1 \rightarrow_{\tilde{\lambda}} e'$ and $e_2 \rightarrow_{\tilde{\lambda}} e'$.

The proof is by mutual induction on c and e . Cases:

- *Step at the root for $c \rightarrow_{\tilde{\lambda}} c_1$ and Step on a v -environment for $c \rightarrow_{\tilde{\lambda}} c_2$, i.e. $c := \langle \lambda x.c_0 \mid v.e \rangle \rightarrow_{\tilde{\lambda}} \langle v \mid (\tilde{\mu}x.c_0)@e \rangle =: c_1$ and $c \rightarrow_{\tilde{\lambda}} \langle \lambda x.c_0 \mid v.e' \rangle =: c_2$ with $e \rightarrow_{\tilde{\lambda}} e'$. Then, $c_2 \rightarrow_{\tilde{\lambda}} \langle v \mid (\tilde{\mu}x.c_0)@e' \rangle =: c'$. According to the co-substitution lemma (Lemma 38), $c_1 \rightarrow_{\tilde{\lambda}} c'$.*
- *Step on an environment for both $c \rightarrow_{\tilde{\lambda}} c_1$ and $c \rightarrow_{\tilde{\lambda}} c_2$, i.e. $c := \langle v \mid e \rangle \rightarrow_{\tilde{\lambda}} \langle v \mid e_1 \rangle =: c_1$ and $c \rightarrow_{\tilde{\lambda}} \langle v \mid e_2 \rangle =: c_2$ with $e_1 \tilde{\lambda} \leftarrow e \rightarrow_{\tilde{\lambda}} e_2$. By *i.h.*, there is an environment e' such that $e_1 \rightarrow_{\tilde{\lambda}} e' \tilde{\lambda} \leftarrow e_2$. According to Remark 36, $c_1 \rightarrow_{\tilde{\lambda}} c' \tilde{\lambda} \leftarrow c_2$ by taking $c' := \langle v \mid e' \rangle$.*
- *Step on a $\tilde{\mu}$ -environment for both $e \rightarrow_{\tilde{\lambda}} e_1$ and $e \rightarrow_{\tilde{\lambda}} e_2$, i.e. $e := \tilde{\mu}\alpha.c \rightarrow_{\tilde{\lambda}} \tilde{\mu}\alpha.c_1 =: e_1$ and $e \rightarrow_{\tilde{\lambda}} \tilde{\mu}\alpha.c_2 =: e_2$ with $c_1 \tilde{\lambda} \leftarrow c \rightarrow_{\tilde{\lambda}} c_2$. By *i.h.*, there is a command c' such that $c_1 \rightarrow_{\tilde{\lambda}} c' \tilde{\lambda} \leftarrow c_2$. So, $e_1 \rightarrow_{\tilde{\lambda}} e' \tilde{\lambda} \leftarrow e_2$ by taking $e' := \tilde{\mu}\alpha.c'$, according to Definition 35.*
- *Step on an environment for both $e \rightarrow_{\tilde{\lambda}} e_1$ and $e \rightarrow_{\tilde{\lambda}} e_2$, i.e. $e := v.e' \rightarrow_{\tilde{\lambda}} v.e'_1 =: e_1$ and $e \rightarrow_{\tilde{\lambda}} v.e'_2 =: e_2$ with $e'_1 \tilde{\lambda} \leftarrow e' \rightarrow_{\tilde{\lambda}} e'_2$. By *i.h.*, there is an environment e'_0 such that $e'_1 \rightarrow_{\tilde{\lambda}} e'_0 \tilde{\lambda} \leftarrow e'_2$. According to Remark 36, $e_1 \rightarrow_{\tilde{\lambda}} e_0 \tilde{\lambda} \leftarrow e_2$ by taking $e_0 := v.e'_0$.*

2. The proof of strong normalization of $\rightarrow_{\bar{\mu}}$ is in [16].

Concerning the proof of strong confluence of $\rightarrow_{\bar{\mu}}$, we prove that:

- (a) (commands) if $c \rightarrow_{\bar{\mu}} c_1$ and $c \rightarrow_{\bar{\mu}} c_2$ with $c_1 \neq c_2$, then there exists c' such that $c_1 \rightarrow_{\bar{\mu}} c'$ and $c_2 \rightarrow_{\bar{\mu}} c'$;
- (b) (environments) if $e \rightarrow_{\bar{\mu}} e_1$ and $e \rightarrow_{\bar{\mu}} e_2$ with $e_1 \neq e_2$, then there exists c' such that $e_1 \rightarrow_{\bar{\mu}} e'$ and $e_2 \rightarrow_{\bar{\mu}} e'$.

The proof is by mutual induction on c and e . Cases:

- *Step at the root for $c \rightarrow_{\bar{\mu}} c_1$ and Step on a $\bar{\mu}$ -environment for $c \rightarrow_{\bar{\mu}} c_2$, i.e. $c := \langle v \mid \bar{\mu}x.c_0 \rangle \rightarrow_{\bar{\mu}} c_0\{x \leftarrow v\} =: c_1$ and $c \rightarrow_{\bar{\mu}} \langle v \mid \bar{\mu}x.c'' \rangle =: c_2$ with $c_0 \rightarrow_{\bar{\mu}} c''$. Then, $c_2 \rightarrow_{\bar{\mu}} c''\{x \leftarrow v\} =: c'$. According to the substitution lemma (Lemma 37), $c_1 \rightarrow_{\bar{\mu}} c'$.*
- *Step on an environment for both $c \rightarrow_{\bar{\mu}} c_1$ and $c \rightarrow_{\bar{\mu}} c_2$, i.e. $c := \langle v \mid e \rangle \rightarrow_{\bar{\mu}} \langle v \mid e_1 \rangle =: c_1$ and $c \rightarrow_{\bar{\mu}} \langle v \mid e_2 \rangle =: c_2$ with $e_1 \bar{\mu} \leftarrow e \rightarrow_{\bar{\mu}} e_2$. By i.h., there is an environment e' such that $e_1 \rightarrow_{\bar{\mu}} e' \bar{\mu} \leftarrow e_2$. According to Remark 36, $c_1 \rightarrow_{\bar{\mu}} c' \bar{\mu} \leftarrow c_2$ by taking $c' := \langle v \mid e' \rangle$.*
- *Step on a $\bar{\mu}$ -environment for both $e \rightarrow_{\bar{\mu}} e_1$ and $e \rightarrow_{\bar{\mu}} e_2$, i.e. $e := \bar{\mu}\alpha.c \rightarrow_{\bar{\mu}} \bar{\mu}\alpha.c_1 =: e_1$ and $e \rightarrow_{\bar{\mu}} \bar{\mu}\alpha.c_2 =: e_2$ with $c_1 \bar{\mu} \leftarrow c \rightarrow_{\bar{\mu}} c_2$. By i.h., there is a command c' such that $c_1 \rightarrow_{\bar{\mu}} c' \bar{\mu} \leftarrow c_2$. So, $e_1 \rightarrow_{\bar{\mu}} e' \bar{\mu} \leftarrow e_2$ by taking $e' := \bar{\mu}\alpha.c'$.*
- *Step on a environment for both $e \rightarrow_{\bar{\mu}} e_1$ and $e \rightarrow_{\bar{\mu}} e_2$, i.e. $e := v \cdot e' \rightarrow_{\bar{\mu}} v \cdot e'_1 =: e_1$ and $e \rightarrow_{\bar{\mu}} v \cdot e'_2 =: e_2$ with $e'_1 \bar{\mu} \leftarrow e' \rightarrow_{\bar{\mu}} e'_2$. By i.h., there is an environment e'_0 such that $e'_1 \rightarrow_{\bar{\mu}} e'_0 \bar{\mu} \leftarrow e'_2$. According to Remark 36, $e_1 \rightarrow_{\bar{\mu}} e_0 \bar{\mu} \leftarrow e_2$ by taking $e_0 := v \cdot e'_0$.*

3. We prove that

- (a) (commands) if $c \rightarrow_{\bar{\mu}} c_1$ and $c \rightarrow_{\bar{\lambda}} c_2$ then $c_1 \neq c_2$ and there exists c' such that $c_1 \rightarrow_{\bar{\lambda}} c'$ and $c_2 \rightarrow_{\bar{\mu}} c'$;
- (b) (environments) if $e \rightarrow_{\bar{\mu}} e_1$ and $e \rightarrow_{\bar{\lambda}} e_2$ then $e_1 \neq e_2$ and there exists c' such that $e_1 \rightarrow_{\bar{\lambda}} e'$ and $e_2 \rightarrow_{\bar{\mu}} e'$.

The proof is by mutual induction on c and e (the proof that $c_1 \neq c_2$ and $e_1 \neq e_2$ is left to the reader). Cases:

- *Step at the root for $c \rightarrow_{\bar{\mu}} c_1$ and Step on a $\bar{\mu}$ -environment for $c \rightarrow_{\bar{\lambda}} c_2$, i.e. $c := \langle v \mid \bar{\mu}x.c_0 \rangle \rightarrow_{\bar{\mu}} c_0\{x \leftarrow v\} =: c_1$ and $c \rightarrow_{\bar{\lambda}} \langle v \mid \bar{\mu}x.c'' \rangle =: c_2$ with $c_0 \rightarrow_{\bar{\lambda}} c''$. Then, $c_2 \rightarrow_{\bar{\mu}} c''\{x \leftarrow v\} =: c'$. By substitution lemma (Lemma 37), $c_1 \rightarrow_{\bar{\lambda}} c'$.*
- *Step on a v -environment for $c \rightarrow_{\bar{\mu}} c_1$ and Step at the root for $c \rightarrow_{\bar{\lambda}} c_2$, i.e. $c := \langle \lambda x.c_0 \mid v \cdot e \rangle \rightarrow_{\bar{\mu}} \langle \lambda x.c_0 \mid v \cdot e' \rangle =: c_1$ with $e \rightarrow_{\bar{\mu}} e'$, and $c \mapsto_{\bar{\lambda}} \langle v \mid (\bar{\mu}x.c_0) @ e \rangle =: c_2$. Then, $c_1 \rightarrow_{\bar{\lambda}} \langle v \mid (\bar{\mu}x.c_0) @ e' \rangle =: c'$ and, by append lemma (Lemma 38), $c_2 \rightarrow_{\bar{\mu}} c'$.*
- *Step on an environment for both $c \rightarrow_{\bar{\mu}} c_1$ and $c \rightarrow_{\bar{\lambda}} c_2$, i.e. $c := \langle v \mid e \rangle \rightarrow_{\bar{\mu}} \langle v \mid e' \rangle =: c_1$ and $c \rightarrow_{\bar{\lambda}} \langle v \mid e'' \rangle =: c_2$, with $e \rightarrow_{\bar{\mu}} e'$ and $e \rightarrow_{\bar{\lambda}} e''$. By i.h., there exists an environment e_0 such that $e' \rightarrow_{\bar{\lambda}} e_0 \bar{\mu} \leftarrow e''$, and hence $c_1 \rightarrow_{\bar{\lambda}} c' \bar{\mu} \leftarrow c_2$ by taking $c' := \langle v \mid e_0 \rangle$, according to Remark 36.*
- *Step on a $\bar{\mu}$ -environment for both $e \rightarrow_{\bar{\mu}} e_1$ and $e \rightarrow_{\bar{\lambda}} e_2$, i.e. $e := \bar{\mu}x.c \rightarrow_{\bar{\mu}} \bar{\mu}x.c_1 =: e_1$ and $e \rightarrow_{\bar{\lambda}} \bar{\mu}x.c_2 =: e_2$, with $c \rightarrow_{\bar{\mu}} c_1$ and $c \rightarrow_{\bar{\lambda}} c_2$. By i.h., there exists a command c_0 such that $c_1 \rightarrow_{\bar{\lambda}} c_0 \bar{\mu} \leftarrow c_2$, and hence $e_1 \rightarrow_{\bar{\lambda}} e' \bar{\mu} \leftarrow e_2$ by taking $e' := \langle v \mid c_0 \rangle$, according to Definition 35.*

- *Step on a v -environment for both $e \rightarrow_{\bar{\mu}} e_1$ and $e \rightarrow_{\bar{\lambda}} e_2$, i.e. $e := v \cdot e_0 \rightarrow_{\bar{\mu}} v \cdot e_{01} =: e_1$ and $e \rightarrow_{\bar{\lambda}} v \cdot e_{02} =: e_2$, with $e_0 \rightarrow_{\bar{\mu}} e_{01}$ and $e_0 \rightarrow_{\bar{\lambda}} e_{02}$. By i.h., there exists an environment e'_0 such that $e_{01} \rightarrow_{\bar{\lambda}} e'_0 \bar{\mu} \leftarrow e_{02}$, and hence $e_1 \rightarrow_{\bar{\lambda}} e' \bar{\mu} \leftarrow e_2$ by taking $e' := v \cdot e'_0$, according to Definition 35.*
- 4. It follows immediately from strong confluence of $\rightarrow_{\bar{\lambda}}$ and $\rightarrow_{\bar{\mu}}$ (Prop. 7.1), strong commutation of $\rightarrow_{\bar{\lambda}}$ and $\rightarrow_{\bar{\mu}}$ (Prop. 7.2) and Hindley-Rosen (Lemma 26). \square

B.2 Proofs of Section 3 (Quantitative Equivalences of λ_{fire} , λ_{shuf} and λ_{vsub})

Proofs of Subsection 3.1 (Equivalence of λ_{fire} and λ_{vsub})

Remark 40. Let $t, u \in \Lambda_{\text{vsub}}$.

1. If $t \equiv u$ then $t \downarrow = u \downarrow$.
2. If $t \equiv u$ then $t \not\rightarrow_{\text{vsub}} u$ (in particular, $t \not\rightarrow_{\text{m}} u$ and $t \not\rightarrow_{\text{e}} u$).

See p. 14

Lemma 8 (Simulation of a \rightarrow_{β_f} -Step by $\rightarrow_{\text{vsub}}$). *Let $t, u \in \Lambda$.*

1. *If $t \rightarrow_{\beta_\lambda} u$ then $t \rightarrow_{\text{m}} \rightarrow_{\text{e}} u$.*
2. *If $t \rightarrow_{\beta_i} u$ then $t \rightarrow_{\text{m}} \equiv s$, with $s \in \Lambda_{\text{vsub}}$ clean and $s \downarrow = u$.*

Proof. Both proofs are by induction on the rewriting step.

1. According to the definition of $t \rightarrow_{\beta_\lambda} u$, there are three cases:
 - *Step at the root, i.e. $t = (\lambda x.s)(\lambda y.r) \mapsto_{\beta_\lambda} s\{x \leftarrow \lambda y.r\} = u$: so, $t \rightarrow_{\text{m}} s[x \leftarrow \lambda y.r] \rightarrow_{\text{e}} u$.*
 - *Application Left, i.e. $t = sr \rightarrow_{\beta_\lambda} s'r = u$ with $s \rightarrow_{\beta_\lambda} s'$: by i.h., $s \rightarrow_{\text{m}} \rightarrow_{\text{e}} s'$ and hence $t = sr \rightarrow_{\text{m}} \rightarrow_{\text{e}} s'r = u$.*
 - *Application Right, i.e. $t = sr \rightarrow_{\beta_\lambda} sr' = u$ with $r \rightarrow_{\beta_\lambda} r'$: by i.h., $r \rightarrow_{\text{m}} \rightarrow_{\text{e}} r'$ and hence $t = sr \rightarrow_{\text{m}} \rightarrow_{\text{e}} sr' = u$.*
2. According to the definition of $t \rightarrow_{\beta_i} u$, there are three cases:
 - *Step at the root, i.e. $t = (\lambda x.s)i \mapsto_{\beta_i} s\{x \leftarrow i\} = u$: then, $t \rightarrow_{\text{m}} s[x \leftarrow i]$ where $s[x \leftarrow i]$ is clean (since $s \in \Lambda$) and $s[x \leftarrow i] \downarrow = s \downarrow \{x \leftarrow i \downarrow\} = u$ ($s \downarrow = s$ and $i \downarrow = i$ because $s, i \in \Lambda$). We conclude since \equiv is reflexive.*
 - *Application Left, i.e. $t = sr \rightarrow_{\beta_i} s'r = u$ with $s \rightarrow_{\beta_i} s'$: by i.h., $s \rightarrow_{\text{m}} \equiv q$ where q is a clean vsub -term such that $q \downarrow = s'$. So, $q = q_0[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$ where $q_0 \in \Lambda$ and i_1, \dots, i_n are inert terms (for some $n \in \mathbb{N}$), moreover we can suppose without loss of generality that $\{x_1, \dots, x_n\} \cap \text{fv}(r) = \emptyset$. Let $u' = (q_0 r)[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$: then, u' is a clean vsub -term such that $qr \equiv u'$ and, according to Remark 40.1, $u' \downarrow = (qr) \downarrow = q \downarrow r \downarrow = s'r = u$. Hence, $t = sr \rightarrow_{\text{m}} \equiv qr \equiv u'$ and we conclude since \equiv is transitive.*
 - *Application Right, i.e. $t = sr \rightarrow_{\beta_i} sr' = u$ with $r \rightarrow_{\beta_i} r'$. Identical to the application left case, just switch left and right.* \square

Lemma 41 (Fireballs are Closed Under Anti-Substitution of Inert Terms).

Let t be a vsub -term and i be an inert term.

1. If $t\{x \leftarrow i\}$ is an abstraction then t is an abstraction.
2. If $t\{x \leftarrow i\}$ is an inert term then t is an inert term;
3. If $t\{x \leftarrow i\}$ is a fireball then t is a fireball.

Proof.

1. If $t\{x \leftarrow i\} = \lambda y.s$ then there is r such that $s = r\{x \leftarrow i\}$, that is $t\{x \leftarrow i\} = \lambda y.(r\{x \leftarrow i\}) = (\lambda y.r)\{x \leftarrow i\}$ and so $t = \lambda y.r$ is an abstraction;
2. By induction on the inert structure of $t\{x \leftarrow i\}$. Cases:
 - *Variable*, i.e. $t\{x \leftarrow i\} = y$, possibly with $x = y$. Then $t = x$ or $t = y$, and in both cases t is inert.
 - *Compound Inert*, i.e. $t\{x \leftarrow i\} = i'f$. If t is a variable then it is inert. Otherwise it is an application $t = us$, and so $u\{x \leftarrow i\} = i'$ and $s\{x \leftarrow i\} = f$. By *i.h.*, u is an inert term. Consider f . Two cases:
 - (a) f is an abstraction. Then by Point 1 s is an abstraction.
 - (b) f is an inert term. Then by *i.h.* s is an inert term.
 In both cases s is a fireball, and so $t = us$ is an inert term.
3. Immediate consequence of Lemmas 41.1-2, since every fireball is either an abstraction or an inert term. \square

Lemma 42 (Substitution of Inert Terms Does Not Create β_f -Redexes).

Let t, u be terms and i be an inert term. There is $s \in \Lambda$ such that:

1. if $t\{x \leftarrow i\} \rightarrow_{\beta_\lambda} u$ then $t \rightarrow_{\beta_\lambda} s$ and $s\{x \leftarrow i\} = u$;
2. if $t\{x \leftarrow i\} \rightarrow_{\beta_i} u$ then $t \rightarrow_{\beta_i} s$ and $s\{x \leftarrow i\} = u$.

Proof. We prove the two points by induction on the evaluation context closing the root redex. Cases:

- *Step at the root*:
 1. *Abstraction Step*, i.e. $t\{x \leftarrow i\} := (\lambda y.r\{x \leftarrow i\})q\{x \leftarrow i\} \mapsto_{\beta_\lambda} r\{x \leftarrow i\}\{y \leftarrow q\{x \leftarrow i\}\} =: u$. By Lemma 41.1, q is an abstraction, since $q\{x \leftarrow i\}$ is an abstraction by hypothesis. Then $t = (\lambda y.r)q \mapsto_{\beta_\lambda} r\{y \leftarrow q\}$. Then $s := r\{x \leftarrow q\}$ verifies the statement, as $s\{x \leftarrow i\} = (r\{y \leftarrow q\})\{x \leftarrow i\} = r\{x \leftarrow i\}\{y \leftarrow q\{x \leftarrow i\}\} = u$.
 2. *Inert Step*, identical to the abstraction subcase, just replace *abstraction* with *inert term* and the use of Lemma 41.1 with the use of Lemma 41.2.
- *Application Left*, i.e. $t = rq$ and reduction takes place in r :
 1. *Abstraction Step*, i.e. $t\{x \leftarrow i\} := r\{x \leftarrow i\}q\{x \leftarrow i\} \rightarrow_{\beta_\lambda} pq\{x \leftarrow i\} =: u$. By *i.h.* there exists $s' \in \Lambda$ such that $p = s'\{x \leftarrow i\}$ and $r \rightarrow_{\beta_\lambda} s'$. Then $s := s'q$ satisfies the statement, as $s\{x \leftarrow i\} = (s'q)\{x \leftarrow i\} = s'\{x \leftarrow i\}q\{x \leftarrow i\} = u$.
 2. *Inert Step*, identical to the abstraction subcase.
- *Application Right*, i.e. $t = rq$ and reduction takes place in q . Identical to the application left case, just switch left and right. \square

Lemma 9 (Projection of a β_f -Step on $\rightarrow_{\text{vsub}}$ via Unfolding). Let t be a clean vsub-term and u be a term.

See p. 14

1. If $t \downarrow \rightarrow_{\beta_\lambda} u$ then $t \rightarrow_{\text{m}} \rightarrow_{\text{e}} s$, with $s \in \Lambda_{\text{vsub}}$ clean s.t. $s \downarrow = u$.
2. If $t \downarrow \rightarrow_{\beta_i} u$ then $t \rightarrow_{\text{m}} \equiv s$, with $s \in \Lambda_{\text{vsub}}$ clean s.t. $s \downarrow = u$.

Proof. Since t is clean, there are a λ -term q and some inert λ -terms i_1, \dots, i_n (with $n \in \mathbb{N}$) such that $t = q[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$. We prove both points by induction on $n \in \mathbb{N}$. The base case (*i.e.* $n = 0$) is given by the simulation of one-step reductions given by Lemma 8, since $t = q \in \Lambda$ and hence $t \downarrow = t$ (recall that, when applying Lemma 8.1, $u \in \Lambda$ implies that u is clean and $u \downarrow = u$).

Consider now $n > 0$. Let $t_{n-1} := q[x_1 \leftarrow i_1] \dots [x_{n-1} \leftarrow i_{n-1}]$: so, $t = t_{n-1}[x_n \leftarrow i_n]$ and $t \downarrow = t_{n-1} \downarrow \{x_n \leftarrow i_n\}$. Both points rely on the fact that the substitution of inert terms cannot create redexes (Lemma 42). Namely,

1. β_λ -step: the application of Lemma 42.1 to $t \downarrow = t_{n-1} \downarrow \{x_n \leftarrow i_n\} \rightarrow_{\beta_\lambda} u$ (since $t_{n-1} \downarrow \in \Lambda$, *i.e.* it has no ES) provides $r \in \Lambda$ such that $t_{n-1} \downarrow \rightarrow_{\beta_\lambda} r$ and $r\{x_n \leftarrow i_n\} = u$. By *i.h.*, $t_n \rightarrow_{\mathbf{m}} \rightarrow_{\mathbf{e}} s$ where s is a clean **vsub**-term such that $s \downarrow = r$, and thus $t = t_{n-1}[x_n \leftarrow i_n] \rightarrow_{\mathbf{m}} \rightarrow_{\mathbf{e}} s[x_n \leftarrow i_n]$. Moreover, $s[x_n \leftarrow i_n]$ is clean and $s[x_n \leftarrow i_n] \downarrow = s \downarrow \{x_n \leftarrow i_n\} = r\{x_n \leftarrow i_n\} = u$.
2. β_i -step: the application of Lemma 42.2 to $t \downarrow = t_{n-1} \downarrow \{x_n \leftarrow i_n\} \rightarrow_{\beta_i} u$ provides $r \in \Lambda$ such that $t_{n-1} \downarrow \rightarrow_{\beta_i} r$ and $r\{x_n \leftarrow i_n\} = u$. By *i.h.*, $t_{n-1} \rightarrow_{\mathbf{m}} \equiv s$ where s is a clean **vsub**-term such that $s \downarrow = r$; thus, $t = t_{n-1}[x_n \leftarrow i_n] \rightarrow_{\mathbf{m}} \equiv s[x_n \leftarrow i_n]$. Moreover, $s[x_n \leftarrow i_n]$ is clean and $s[x_n \leftarrow i_n] \downarrow = s \downarrow \{x_n \leftarrow i_n\} = r\{x_n \leftarrow i_n\} = u$. \square

See p. 14

Lemma 10. *Let t be a clean **vsub**-term. If $t \downarrow$ is a fireball, then t is $\{\mathbf{m}, \mathbf{e}_\lambda\}$ -normal and its body is a fireball.*

Proof. First, we prove that if $t \downarrow$ is a fireball then for some fireball f and inert terms i_1, \dots, i_n one has $t = f[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$. Since t is clean, there are a term u and some inert terms i_1, \dots, i_n (with $n \in \mathbb{N}$) such that $t = u[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$. We prove by induction on $n \in \mathbb{N}$ that u is a fireball.

If $n = 0$, then $t = u \in \Lambda$, thus $u = t \downarrow$ and hence u is a fireball by hypothesis.

Suppose $n > 0$ and let $s := u[x_1 \leftarrow i_1] \dots [x_{n-1} \leftarrow i_{n-1}]$, which is a clean **vsub**-term: then, $t = s[x_n \leftarrow i_n]$ and hence $t \downarrow = s \downarrow \{x_n \leftarrow i_n\}$ (as $i_n \downarrow = i_n$ because $i_n \in \Lambda$). By Lemma 41.3, $s \downarrow$ is a fireball. By *i.h.*, u is a fireball.

So, we have just proved that $t = f[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$ for some fireball f and inert terms i_1, \dots, i_n . Now, fireballs (in particular, inert terms) are **vsub**-normal. Indeed, fireballs are without ES and hence without **e**-redexes, moreover it is easy to prove that fireballs are **m**-normal (by simply adapting the proof of Lemma 29).

Therefore, $t = f[x_1 \leftarrow i_1] \dots [x_n \leftarrow i_n]$ can only have **e_y**-redexes (when some i_k is a variable). \square

See p. 15

Lemma 11 (Linear Postponement of $\rightarrow_{\mathbf{e}_y}$). *Let $t, u, s \in \Lambda_{\mathbf{vsub}}$.*

1. *If $t \rightarrow_{\mathbf{e}_y} s \rightarrow_{\mathbf{m}} u$ then $t \rightarrow_{\mathbf{m}} \rightarrow_{\mathbf{e}_y} u$.*
2. *If $t \rightarrow_{\mathbf{e}_y} \rightarrow_{\mathbf{e}_\lambda} u$ then $t \rightarrow_{\mathbf{e}_\lambda} \rightarrow_{\mathbf{e}} u$.*
3. *If $d: t \rightarrow_{\mathbf{vsub}}^* u$ then $e: t \rightarrow_{\mathbf{m}, \mathbf{e}_\lambda}^* \rightarrow_{\mathbf{e}_y}^* u$ with $|e|_{\mathbf{vsub}} = |d|_{\mathbf{vsub}}$, $|e|_{\mathbf{m}} = |d|_{\mathbf{m}}$, $|e|_{\mathbf{e}} = |d|_{\mathbf{e}}$, and $|e|_{\mathbf{e}_\lambda} \geq |d|_{\mathbf{e}_\lambda}$.*

Proof. 1. By induction on the definition of $t \rightarrow_{\mathbf{e}_y} s$. Since the **e_y**-step cannot create in s new **m**-redexes not occurring in t , the **m**-redex fired in $s \rightarrow_{\mathbf{m}} u$ is (a residual of a **m**-redex) already occurring in t . So, there are the following cases.

- *Step at the Root* for $t \rightarrow_{\text{ey}} s$ and *ES Left* for $s \rightarrow_{\text{m}} u$, i.e. $t := r[z \leftarrow L\langle x \rangle] \rightarrow_{\text{ey}} L\langle r\{z \leftarrow x\} \rangle =: s$ and $s \rightarrow_{\text{m}} L\langle r'\{z \leftarrow x\} \rangle =: u$ with $r \rightarrow_{\text{m}} r'$: then $t \rightarrow_{\text{m}} r'[z \leftarrow L\langle x \rangle] \rightarrow_{\text{ey}} u$;
- *Step at the Root* for $t \rightarrow_{\text{ey}} s$ and *ES “quasi-Right”* for $s \rightarrow_{\text{m}} u$, i.e. $t := r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \rightarrow_{\text{ey}} r\{z \leftarrow x\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: s$ and

$$t \rightarrow_{\text{m}} r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] =: u$$

- for some $n > 0$, and $t_j \rightarrow_{\text{m}} t'_j$ for some $1 \leq j \leq n$: then, $t \rightarrow_{\text{m}} r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] \rightarrow_{\text{ey}} u$;
- *Application Left* for $t \rightarrow_{\text{ey}} s$ and *Application Right* for $s \rightarrow_{\text{m}} u$, i.e. $t := rq \rightarrow_{\text{ey}} r'q =: s$ and $s \rightarrow_{\text{m}} r'q' =: u$ with $r \rightarrow_{\text{ey}} r'$ and $q \rightarrow_{\text{m}} q'$: then, $t \rightarrow_{\text{m}} rq' \rightarrow_{\text{ey}} u$;
 - *Application Left* for both $t \rightarrow_{\text{ey}} s$ and $s \rightarrow_{\text{m}} u$, i.e. $t := rq \rightarrow_{\text{ey}} r'q =: s$ and $s \rightarrow_{\text{m}} r''q =: u$ with $r \rightarrow_{\text{ey}} r'$ and $r' \rightarrow_{\text{m}} r''$: by i.h., $r \rightarrow_{\text{m}} r''$, hence $t \rightarrow_{\text{m}} r''q \rightarrow_{\text{ey}} u$;
 - *Application Left* for $t \rightarrow_{\text{ey}} s$ and *Step at the Root* for $s \rightarrow_{\text{m}} u$, i.e. $t := (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]r \rightarrow_{\text{ey}} (\lambda x.q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]r =: s$ with $n > 0$ and $t_j \rightarrow_{\text{ey}} t'_j$ for some $1 \leq j \leq n$, and

$$s \rightarrow_{\text{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] =: u$$

then,

$$t \rightarrow_{\text{m}} q[x \leftarrow r][x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] \rightarrow_{\text{ey}} u;$$

- *Application Right* for $t \rightarrow_{\text{ey}} s$ and *Application Left* for $s \rightarrow_{\text{m}} u$, i.e. $t := qr \rightarrow_{\text{ey}} qr' =: s$ and $s \rightarrow_{\text{m}} q'r' =: u$ with $r \rightarrow_{\text{ey}} r'$ and $q \rightarrow_{\text{m}} q'$: then, $t \rightarrow_{\text{m}} q'r \rightarrow_{\text{ey}} u$;
- *Application Right* for both $t \rightarrow_{\text{ey}} s$ and $s \rightarrow_{\text{m}} u$, i.e. $t := qr \rightarrow_{\text{ey}} qr' =: s$ and $s \rightarrow_{\text{m}} qr'' =: u$ with $r \rightarrow_{\text{ey}} r'$ and $r' \rightarrow_{\text{m}} r''$: by i.h., $r \rightarrow_{\text{m}} r''$, hence $t \rightarrow_{\text{m}} r''q \rightarrow_{\text{ey}} u$;
- *Application Right* for $t \rightarrow_{\text{ey}} s$ and *Step at the Root* for $s \rightarrow_{\text{m}} u$, i.e. $t := L\lambda x.qr \rightarrow_{\text{ey}} L\langle \lambda x.q \rangle r' =: s$ with $r \rightarrow_{\text{ey}} r'$, and $s \rightarrow_{\text{m}} L\langle q[x \leftarrow r'] \rangle =: u$: then, $t \rightarrow_{\text{m}} L\langle q[x \leftarrow r] \rangle \rightarrow_{\text{ey}} u$;
- *ES Left* for $t \rightarrow_{\text{ey}} s$ and *ES Right* for $s \rightarrow_{\text{m}} u$, i.e. $t := r[x \leftarrow q] \rightarrow_{\text{ey}} r'[x \leftarrow q] =: s$ and $s \rightarrow_{\text{m}} r'[x \leftarrow q'] =: u$ with $r \rightarrow_{\text{ey}} r'$ and $q \rightarrow_{\text{m}} q'$: then, $t \rightarrow_{\text{m}} r[x \leftarrow q'] \rightarrow_{\text{ey}} u$;
- *ES Left* for both $t \rightarrow_{\text{ey}} s$ and $s \rightarrow_{\text{m}} u$, i.e. $t := r[x \leftarrow q] \rightarrow_{\text{ey}} r'[x \leftarrow q] =: s$ and $s \rightarrow_{\text{m}} r''[x \leftarrow q] =: u$ with $r \rightarrow_{\text{ey}} r'$ and $r' \rightarrow_{\text{m}} r''$: by i.h., $r \rightarrow_{\text{m}} r''$, hence $t \rightarrow_{\text{m}} r''[x \leftarrow q] \rightarrow_{\text{ey}} u$;
- *ES Right* for $t \rightarrow_{\text{ey}} s$ and *ES Left* for $s \rightarrow_{\text{m}} u$, i.e. $t := q[x \leftarrow r] \rightarrow_{\text{ey}} q[x \leftarrow r'] =: s$ and $s \rightarrow_{\text{m}} q'[x \leftarrow r'] =: u$ with $r \rightarrow_{\text{ey}} r'$ and $q \rightarrow_{\text{m}} q'$: then, $t \rightarrow_{\text{m}} q'[x \leftarrow r] \rightarrow_{\text{ey}} u$;
- *ES Right* for both $t \rightarrow_{\text{ey}} s$ and $s \rightarrow_{\text{m}} u$, i.e. $t := q[x \leftarrow r] \rightarrow_{\text{ey}} q[x \leftarrow r'] =: s$ and $s \rightarrow_{\text{m}} q[x \leftarrow r''] =: u$ with $r \rightarrow_{\text{ey}} r'$ and $r' \rightarrow_{\text{m}} r''$: by i.h., $r \rightarrow_{\text{m}} r''$, hence $t \rightarrow_{\text{m}} q[x \leftarrow r''] \rightarrow_{\text{ey}} u$.

2. By induction on the definition of $t \rightarrow_{e_y} s$. Since the e_y -step cannot create in s new e_λ -redexes not occurring in t , the e_λ -redex fired in $s \rightarrow_{e_\lambda} u$ is (a residual of a e_λ -redex) already occurring in t . So, there are the following cases.

– *Step at the Root for both $t \rightarrow_{e_y} s$ and $s \rightarrow_{e_\lambda} u$, i.e.*

$$t := r[x \leftarrow L'(z)][y \leftarrow L(\lambda x.q)] \rightarrow_{e_y} L'\langle r\{x \leftarrow z\} \rangle[y \leftarrow L(\lambda x.q)] =: s$$

and $s \rightarrow_{e_\lambda} L\langle L'\langle r\{x \leftarrow z\} \rangle\{y \leftarrow \lambda x.q\} \rangle =: u$ (with possibly $y = z$). We set $L'' := L'\{y \leftarrow \lambda x.q\}$ i.e. L'' is the substitution context obtained from L' by the capture-avoiding substitution of $\lambda x.q$ for each free occurrence of y in L' . We can suppose without loss of generality that $y \notin \text{fv}(L) \cup \text{fv}(r)$. There are two sub-cases:

- either $y = z$ and then $t \rightarrow_{e_\lambda} r[x \leftarrow L\langle L''(\lambda x.q) \rangle] \rightarrow_{e_\lambda} L\langle L''\langle r\{x \leftarrow \lambda x.q\} \rangle \rangle = u$,
 - or $y \neq z$ and then $t \rightarrow_{e_\lambda} r[x \leftarrow L\langle L''(z) \rangle] \rightarrow_{e_y} L\langle L''\langle r\{x \leftarrow z\} \rangle \rangle = u$.
- *Step at the Root for $t \rightarrow_{e_y} s$ and ES Left for $s \rightarrow_{e_\lambda} u$, i.e. $t := r[z \leftarrow L\langle x \rangle] \rightarrow_{e_y} L\langle r\{z \leftarrow x\} \rangle =: s$ and $s \rightarrow_{e_\lambda} L\langle r'\{z \leftarrow x\} \rangle =: u$ with $r \rightarrow_{e_\lambda} r'$: then $t \rightarrow_{e_\lambda} r'[z \leftarrow L\langle x \rangle] \rightarrow_{e_y} u$;*
- *Step at the Root for $t \rightarrow_{e_y} s$ and ES “quasi-Right” for $s \rightarrow_{e_\lambda} u$, i.e. $t := r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \rightarrow_{e_y} r\{z \leftarrow x\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] =: s$ for some $n > 0$, and $t_j \rightarrow_{e_\lambda} t'_j$ for some $1 \leq j \leq n$, and*

$$s \rightarrow_{e_\lambda} r\{z \leftarrow x\}[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] =: u$$

then, $t \rightarrow_{e_\lambda} r[z \leftarrow x[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] \rightarrow_{e_y} u$;

- *Application Left for $t \rightarrow_{e_y} s$ and Application Right for $s \rightarrow_{e_\lambda} u$, i.e. $t := r q \rightarrow_{e_y} r' q =: s$ and $s \rightarrow_{e_\lambda} r' q' =: u$ with $r \rightarrow_{e_y} r'$ and $q \rightarrow_{e_\lambda} q'$: then, $t \rightarrow_{e_\lambda} r q' \rightarrow_{e_y} u$;*
- *Application Left for both $t \rightarrow_{e_y} s$ and $s \rightarrow_{e_\lambda} u$, i.e. $t := r q \rightarrow_{e_y} r' q =: s$ and $s \rightarrow_{e_\lambda} r'' q =: u$ with $r \rightarrow_{e_y} r'$ and $r' \rightarrow_{e_\lambda} r''$: by i.h., $r \rightarrow_{e_\lambda} \rightarrow_e r''$, hence $t \rightarrow_{e_\lambda} \rightarrow_e u$;*
- *Application Right for $t \rightarrow_{e_y} s$ and Application Left for $s \rightarrow_{e_\lambda} u$, i.e. $t := q r \rightarrow_{e_y} q r' =: s$ and $s \rightarrow_{e_\lambda} q' r' =: u$ with $r \rightarrow_{e_y} r'$ and $q \rightarrow_{e_\lambda} q'$: then, $t \rightarrow_{e_\lambda} q' r \rightarrow_{e_y} u$;*
- *Application Right for both $t \rightarrow_{e_y} s$ and $s \rightarrow_{e_\lambda} u$, i.e. $t := q r \rightarrow_{e_y} q r' =: s$ and $s \rightarrow_{e_\lambda} q r'' =: u$ with $r \rightarrow_{e_y} r'$ and $r' \rightarrow_{e_\lambda} r''$: by i.h., $r \rightarrow_{e_\lambda} \rightarrow_e r''$, hence $t \rightarrow_{e_\lambda} \rightarrow_e u$;*
- *ES Left for $t \rightarrow_{e_y} s$ and Step at the Root for $s \rightarrow_{e_\lambda} u$, i.e. $t := r[z \leftarrow L\langle \lambda y.q \rangle] \rightarrow_{e_y} r'[z \leftarrow L\langle \lambda y.q \rangle] =: s$ and $s \rightarrow_{e_\lambda} L\langle r'\{z \leftarrow \lambda y.q\} \rangle =: u$ with $r \rightarrow_{e_y} r'$: this means that in r there is an ES of the form $[y \leftarrow x]$ (possibly $x = z$) which is fired in $r \rightarrow_{e_y} r'$; then, $t \rightarrow_{e_\lambda} L\langle r\{z \leftarrow \lambda y.q\} \rangle \rightarrow_e u$, where the last e-step is a e_λ -step if $x = z$, otherwise it is a e_y -step;*
- *ES Left for $t \rightarrow_{e_y} s$ and ES Right for $s \rightarrow_{e_\lambda} u$, i.e. $t := r[x \leftarrow q] \rightarrow_{e_y} r'[x \leftarrow q] =: s$ and $s \rightarrow_{e_\lambda} r'[x \leftarrow q'] =: u$ with $r \rightarrow_{e_y} r'$ and $q \rightarrow_{e_\lambda} q'$: then, $t \rightarrow_{e_\lambda} r[x \leftarrow q'] \rightarrow_{e_y} u$;*

- *ES Left* for both $t \rightarrow_{e_y} s$ and $s \rightarrow_{e_\lambda} u$, i.e. $t := r[x \leftarrow q] \rightarrow_{e_y} r'[x \leftarrow q] =: s$ and $s \rightarrow_{e_\lambda} r''[x \leftarrow q] =: u$ with $r \rightarrow_{e_y} r'$ and $r' \rightarrow_{e_\lambda} r''$: by i.h., $r \rightarrow_{e_\lambda} \rightarrow_e r''$, so $t \rightarrow_{e_\lambda} \rightarrow_e u$;
- *ES Right* for $t \rightarrow_{e_y} s$ and *Step at the Root* for $s \rightarrow_{e_\lambda} u$, i.e.

$$t := r[z \leftarrow (\lambda y. q)[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n]] \\ \rightarrow_{e_y} r[z \leftarrow (\lambda y. q)[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n]] =: s$$

for some $n > 0$, and $t_j \rightarrow_{e_y} t'_j$ for some $1 \leq j \leq n$, and $s \rightarrow_{e_\lambda} r\{z \leftarrow \lambda y. q\}[x_1 \leftarrow t_1] \dots [x_j \leftarrow t'_j] \dots [x_n \leftarrow t_n] =: u$: then,

$$t \rightarrow_{e_\lambda} r\{z \leftarrow \lambda y. q\}[x_1 \leftarrow t_1] \dots [x_n \leftarrow t_n] \rightarrow_{e_y} u;$$

- *ES Right* for $t \rightarrow_{e_y} s$ and *ES Left* for $s \rightarrow_{e_\lambda} u$, i.e. $t := q[x \leftarrow r] \rightarrow_{e_y} q[x \leftarrow r'] =: s$ and $s \rightarrow_{e_\lambda} q'[x \leftarrow r'] =: u$ with $r \rightarrow_{e_y} r'$ and $q \rightarrow_{e_\lambda} q'$: then, $t \rightarrow_{e_\lambda} q'[x \leftarrow r] \rightarrow_{e_y} u$;
- *ES Right* for both $t \rightarrow_{e_y} s$ and $s \rightarrow_{e_\lambda} u$, i.e. $t := q[x \leftarrow r] \rightarrow_{e_y} q[x \leftarrow r'] =: s$ and $s \rightarrow_{e_\lambda} q[x \leftarrow r''] =: u$ with $r \rightarrow_{e_y} r'$ and $r' \rightarrow_{e_\lambda} r''$: by i.h., $r \rightarrow_{e_\lambda} \rightarrow_e r''$, hence $t \rightarrow_{e_\lambda} \rightarrow_e u$.

3. By induction on $|d|_{\text{vsub}} \in \mathbb{N}$, using Lemmas 11.1-2 in the inductive case. \square

Theorem 12 (Quantitative Simulation of λ_{fire} in λ_{vsub}). *Let $t, u \in \Lambda$. If $d: t \rightarrow_{\beta_f}^* u$ then there are $s, r \in \Lambda_{\text{vsub}}$ and $e: t \rightarrow_{\text{vsub}}^* r$ such that* See p. 15

1. Qualitative Relationship: $r \equiv s$, $u = s \downarrow = r \downarrow$ and s is clean;
2. Quantitative Relationship:
 - (a) Multiplicative Steps: $|d|_{\beta_f} = |e|_{\text{m}}$;
 - (b) Exponential (Abstraction) Steps: $|d|_{\beta_\lambda} = |e|_{e_\lambda} = |e|_e$.
3. Normal Forms: if u is β_f -normal then there exists $f: r \rightarrow_{e_y}^* q$ such that q is a vsub -normal form and $|f|_{e_y} \leq |e|_{\text{m}} - |e|_{e_\lambda}$.

Proof. The first two points are proved together.

- 1-2. By the remark at the beginning of this section of the Appendix (Remarks 40.1-2), it is sufficient to show that there exists $e: t \rightarrow_{\text{vsub}}^* s \in \Lambda_{\text{vsub}}$ such that $u = s \downarrow$ with s clean, and $|d|_{\beta_f} = |e|_{\text{m}}$ and $|d|_{\beta_\lambda} = |e|_{e_\lambda}$ (the fact that $|e|_{e_\lambda} = |e|_e$ is immediate, since the simulation obtained by iterating the projection in Lemma 9 never uses \rightarrow_{e_y}). We proceed by induction on $|d|_{\beta_f} \in \mathbb{N}$. Cases:

- *Empty derivation*, i.e. $|d|_{\beta_f} = 0$ then $t = u$ and $|d|_{\beta_\lambda} = 0$, so we conclude taking $s := u$ and e as the empty derivation.
- *Non-empty derivation*, i.e. $|d|_{\beta_f} > 0$: then, $d: t \rightarrow_{\beta_f}^* r \rightarrow_{\beta_f} u$ and let $d': t \rightarrow_{\beta_f}^* r$ be the derivation obtained from d by removing its last step $r \rightarrow_{\beta_f} u$. By i.h., there is $e': t \rightarrow_{\text{vsub}}^* q$ such that $r = q \downarrow$, q is clean, $|d'|_{\beta_f} = |e'|_{\text{m}}$, and $|d'|_{\beta_\lambda} = |e'|_{e_\lambda}$. By applying Lemma 9 to the last step $r \rightarrow_{\beta_f} u$ of d , we obtain s such that either $q \rightarrow_{\text{m}} \rightarrow_e s$, if $q \downarrow \rightarrow_{\beta_v} u$, or $q \rightarrow_{\text{m}} \equiv s$, if $q \downarrow \rightarrow_{\beta_i} u$, and in both cases s is a clean vsub -term such that

$s' \downarrow = u$. Note that both cases can be summed up with $q \rightarrow_m \xrightarrow{\equiv}_e s$. Composing the two obtained derivations $e': t \rightarrow_{\text{vsub}}^* q$ and $q \rightarrow_m \xrightarrow{\equiv}_e s$, we obtain the derivation $e'': t \rightarrow_{\text{vsub}}^* q \rightarrow_m \xrightarrow{\equiv}_e s$ that satisfies the quantitative relationships but not yet the qualitative one, as \equiv appears between two steps of e'' . It is then enough to apply the strong bisimulation property of \equiv (Lemma 5.2), that provides a derivation $e: t \rightarrow_{\text{vsub}}^* \rightarrow_m \xrightarrow{\equiv}_e s$ with the same quantitative properties of e'' .

3. If u is β_f -normal then it is a fireball (by open harmony, Prop. 2) and so s is $\{\mathbf{m}, \mathbf{e}_\lambda\}$ -normal by Lemma 10. By Prop. 4.1, $\rightarrow_{\mathbf{e}_y}$ terminates and so there are p and a derivation $g: s \rightarrow_{\mathbf{e}_y}^* p$ such that p is a \mathbf{e}_y -normal form. If p is not a vsub -normal form, then it has a $\{\mathbf{m}, \mathbf{e}_\lambda\}$ -redex, but by postponement of $\rightarrow_{\mathbf{e}_y}$ (Lemma 11) such a redex was already in s , against hypothesis. So p is a vsub -normal form. Then we have $r \equiv s \rightarrow_{\mathbf{e}_y}^* p$. Postponing \equiv (Lemmas 5.2-3), we obtain that there exists a vsub -normal form q and a derivation $f: r \rightarrow_{\mathbf{e}_y}^* q \equiv p$.

To estimate the length of f consider e followed by f , i.e. $e; f: t \rightarrow_{\mathbf{m}, \mathbf{e}_\lambda}^* r \rightarrow_{\mathbf{e}_y}^* q$. By Prop. 4.4, $|e; f|_{\mathbf{e}} \leq |e; f|_{\mathbf{m}} = |e|_{\mathbf{m}}$, and since $|e; f|_{\mathbf{e}} = |e; f|_{\mathbf{e}_\lambda} + |e; f|_{\mathbf{e}_y} = |e|_{\mathbf{e}_\lambda} + |f|_{\mathbf{e}_y}$ we obtain $|e|_{\mathbf{e}_\lambda} + |f|_{\mathbf{e}_y} \leq |e|_{\mathbf{m}}$, i.e. $|f|_{\mathbf{e}_y} \leq |e|_{\mathbf{m}} - |e|_{\mathbf{e}_\lambda}$. \square

See p. 15

Corollary 13 (Linear Termination Equivalence of λ_{vsub} and λ_{fire}). *Let $t \in \Lambda$. There exists a β_f -normalizing derivation d from t iff there exists a vsub -normalizing derivation e from t . Moreover, $|d|_{\beta_f} \leq |e|_{\text{vsub}} \leq 2|d|_{\beta_f}$, i.e. they are linearly related.*

Proof.

- \Rightarrow : Let $d: t \rightarrow_{\beta_f}^* u$ be a β_f -normalizing derivation and $e: t \rightarrow_{\text{vsub}}^* \rightarrow_{\mathbf{e}_y}^* q$ be the composition of its projection in λ_{vsub} with the extension to a \mathbf{e}_y -derivation with q vsub -normal, according to Thm. 12. Then e is a vsub -normalizing derivation from t .
- \Leftarrow : By contradiction, suppose that there is a diverging β_f -derivation from t in λ_{fire} . By Thm. 12 it projects to a vsub -derivation in λ_{vsub} that is at least as long as the one in λ_{fire} , absurd.

About lengths, $|d|_{\beta_f} \leq |e|_{\text{vsub}}$ since $|e|_{\mathbf{m}} = |d|_{\beta_f}$ (Thm. 12.2). By Prop. 4.4, $|e|_{\mathbf{e}} \leq |e|_{\mathbf{m}}$ and so $|e|_{\text{vsub}} = |e|_{\mathbf{m}} + |e|_{\mathbf{e}} \leq 2|d|_{\beta_f}$. \square

Proofs of Subsection 3.2 (Equivalence of λ_{shuf} and λ_{vsub})

Lemma 43 (Simulation of a shuf-Step on λ_{vsub}). *Let $t, u \in \Lambda$.*

1. *If $t \rightarrow_{\sigma^b} u$ then there exist $s, r \in \Lambda_{\text{vsub}}$ s.t. $t \rightarrow_{\mathbf{m}}^+ s \equiv r \xrightarrow{\mathbf{m}}^+ u$.*
2. *If $t \rightarrow_{\beta^b} u$ then there exists $s \in \Lambda_{\text{vsub}}$ s.t. $t \rightarrow_{\mathbf{m}}^+ \rightarrow_{\mathbf{e}}^* s \xrightarrow{\mathbf{m}}^+ u$.*

Proof. 1. By induction on the definition of $t \rightarrow_{\sigma^b} s$, following Remark 33. There are four cases:

- (a) *Step at the root, i.e. $t \mapsto_{\sigma} u$.*

- i. either $t := (\lambda x.q)sr \mapsto_{\sigma_1} (\lambda x.qr)s =: u$ with $x \notin \text{fv}(r)$, and then $t = (\lambda x.q)sr \rightarrow_m q[x \leftarrow s]r \equiv (qr)[x \leftarrow s]_m \leftarrow (\lambda x.qr)s = u$;
- ii. or $t := v((\lambda x.s)r) \mapsto_{\sigma_3} (\lambda x.vs)r =: u$ with $x \notin \text{fv}(v)$ and then $t = v((\lambda x.s)r) \rightarrow_m v(s[x \leftarrow r]) \equiv (vs)[x \leftarrow r]_m \leftarrow (\lambda x.vs)r = u$.
- (b) *Application Left*, i.e. $t := sr \rightarrow_{\sigma^b} qr =: u$ with $s \rightarrow_{\sigma^b} q$. The result follows by the *i.h.*, as \rightarrow_m and \equiv are closed by applicative contexts.
- (c) *Application Right*, i.e. $t := sr \rightarrow_{\sigma^b} sq =: u$ with $r \rightarrow_{\sigma^b} q$. The result follows by the *i.h.*, as \rightarrow_m and \equiv are closed by applicative contexts.
- (d) *Inside a β -context*, i.e. $t := (\lambda x.s)r \rightarrow_{\sigma^b} (\lambda x.q)r =: u$ with $s \rightarrow_{\sigma^b} q$. By *i.h.*, $s \rightarrow_m^+ s' \equiv q'_m \leftarrow q$. Now, \rightarrow_m and \equiv are not closed by balanced contexts, but it is enough to apply a further \rightarrow_m step to the balanced context (as \rightarrow_m and \equiv are instead closed by substitution contexts), obtaining $t = (\lambda x.s)r \rightarrow_m s[x \leftarrow r] \rightarrow_m^+ s'[x \leftarrow r] \equiv q'[x \leftarrow r]_m \leftarrow q[x \leftarrow r]_m \leftarrow (\lambda x.q)r = u$.
- 2. By induction on the definition of $t \rightarrow_{\beta_v^b} u$, there are four cases:
 - (a) *Step at the root*, i.e. $t = (\lambda x.r)v \mapsto_{\beta_v} r\{x \leftarrow v\} = u$. So, $t \rightarrow_m r[x \leftarrow v] \rightarrow_e u$.
 - (b) *Application Left*. It follows by the *i.h.*, as \rightarrow_m and \rightarrow_e are closed by applicative contexts.
 - (c) *Application Right*. It follows by the *i.h.*, as \rightarrow_m and \rightarrow_e are closed by applicative contexts.
 - (d) *Step inside a β -context*, i.e. $t = (\lambda x.s)r \rightarrow_{\beta_v^b} (\lambda x.q)r = u$ with $s \rightarrow_{\beta_v^b} q$. By *i.h.*, $s \rightarrow_m^+ \rightarrow_e p_m^* \leftarrow q$. Now, \rightarrow_m and \rightarrow_e are not closed by balanced contexts, but it is enough to apply a further \rightarrow_m step to the balanced context (as \rightarrow_m and \rightarrow_e are instead closed by substitution contexts), obtaining $(\lambda x.s)r \rightarrow_m s[x \leftarrow r] \rightarrow_m^+ \rightarrow_e p[x \leftarrow r]_m^* \leftarrow q[x \leftarrow r]_m \leftarrow (\lambda x.q)r$. \square

Lemma 14 (Projecting a shuf-Step on $\rightarrow_{\text{vsub} \equiv}$ via m-nf). *Let $t, u \in \Lambda$.*

See p. 15

- 1. *If $t \rightarrow_{\sigma^b} u$ then $\text{m}(t) \equiv \text{m}(u)$.*
- 2. *If $t \rightarrow_{\beta_v^b} u$ then $\text{m}(t) \rightarrow_e \rightarrow_m^* \text{m}(u)$.*

Proof.

- 1. By Lemma 43.1 there exist $s, r \in \Lambda_{\text{vsub}}$ s.t. $t \rightarrow_m^+ s \equiv r_m^* \leftarrow u$. By existence and uniqueness of the m -normal form (Propositions 4.1-1 and Prop. 25.1), $s \rightarrow_m^+ \text{m}(s) = \text{m}(t)$. By Lemma 5.2, there is $q \in \Lambda_{\text{vsub}}$ s.t. $r \rightarrow_m^+ q \equiv \text{m}(t)$. By Lemma 5.3, q is m -normal; in particular, $q = \text{m}(r) = \text{m}(u)$ according to Prop. 25.1. Thus, $\text{m}(t) \equiv q = \text{m}(u)$.
- 2. By Lemma 43.2 there are $s, r \in \Lambda_{\text{vsub}}$ such that $t \rightarrow_m^+ s \rightarrow_e r_m^* \leftarrow u$. By existence and uniqueness of the m -normal form (Propositions 4.1-1 and Prop. 25.1), $\text{m}(s) = \text{m}(t)$. As $\text{m}(t)_m^* \leftarrow s \rightarrow_e r$, there is $q \in \Lambda_{\text{vsub}}$ s.t. $\text{m}(t) \rightarrow_e q_m^* \leftarrow r$ according to strong commutation of \rightarrow_m and \rightarrow_e (Prop. 4.2). Thus, $\text{m}(t) \rightarrow_e q_m^* \leftarrow u$ and hence $\text{m}(t) \rightarrow_e \rightarrow_m^* \text{m}(u)$ since $\text{m}(u) = \text{m}(q)$ by Prop. 25.1. \square

Lemma 15 (Projection Preserves Normal Forms). *Let $t \in \Lambda$. If t is shuf-normal then $\text{m}(t)$ is vsub-normal.*

See p. 16

Proof. In [8, Prop. 12] (where the reduction $\rightarrow_{\text{shuf}}$ is denoted by \rightarrow_w) it has been shown that:

1. a term is **shuf**-normal iff it is of the form w ,
2. a term is **shuf**-normal and is neither a value nor a β -redex (*i.e.* of the form $(\lambda x.t)u$) iff it is of the form a ,

where the forms w and a are defined by mutual induction as follows:

$$a ::= xv \mid xa \mid aw \qquad w ::= v \mid a \mid (\lambda x.w)a.$$

The idea is the following: on the one hand, not only terms of the form a are not values but also they cannot reduce to value through **m**-derivations; on the other hand, any **m**-derivation from a term of the form w cannot create an ES of the form $[x \leftarrow L\langle v \rangle]$, therefore the **e**-normality of w (which is without ES) is preserved in its **m**-normal form $\mathbf{m}(w)$ and hence $\mathbf{m}(w)$ is **vsub**-normal.

More formally, consider the types a_{vsub} and w_{vsub} of **vsub**-terms defined by mutual induction as follows (v is a value, without ES):

$$\begin{aligned} a_{\text{vsub}} &::= xv \mid xa_{\text{vsub}} \mid a_{\text{vsub}}w_{\text{vsub}} \\ w_{\text{vsub}} &::= v \mid a_{\text{vsub}} \mid w_{\text{vsub}}[x \leftarrow a_{\text{vsub}}]. \end{aligned}$$

First, we prove by mutual induction on a and w that the **m**-normal form $\mathbf{m}(a)$ of a is of the form a_{vsub} , and the **m**-normal form $\mathbf{m}(w)$ of w is of the form w_{vsub} . The base cases are $\mathbf{m}(v) = v$ (since $\rightarrow_{\mathbf{m}}$ does not reduce under λ 's) and $\mathbf{m}(xv) = xv$. Inductive cases:

1. $\mathbf{m}(xa) = x\mathbf{m}(a) = xa_{\text{vsub}}$ where $\mathbf{m}(a) = a_{\text{vsub}}$ by *i.h.*,
2. $\mathbf{m}(aw) = \mathbf{m}(a)\mathbf{m}(w) = a_{\text{vsub}}w_{\text{vsub}}$ (since a_{vsub} is not an abstraction) where $\mathbf{m}(a) = a_{\text{vsub}}$ and $\mathbf{m}(w) = w_{\text{vsub}}$ by *i.h.*,
3. $\mathbf{m}((\lambda x.w)a) = \mathbf{m}(w)[x \leftarrow \mathbf{m}(a)] = w_{\text{vsub}}[x \leftarrow a_{\text{vsub}}]$ (since a_{vsub} is not of the form $L\langle v \rangle$) where $\mathbf{m}(a) = a_{\text{vsub}}$ and $\mathbf{m}(w) = w_{\text{vsub}}$ by *i.h.*

To conclude the proof of Lemma 15, it is sufficient to observe that all terms of type w_{vsub} are **vsub**-normal, see [3, Lemma 5] (where $\rightarrow_{\text{vsub}}$ is denoted by \rightarrow_w). \square

See p. 16

Theorem 16 (Quantitative Simulation of λ_{shuf} in λ_{vsub}). *Let $t, u \in \Lambda$. If $d: t \rightarrow_{\text{shuf}}^* u$ then there are $s \in \Lambda_{\text{vsub}}$ and $e: t \rightarrow_{\text{vsub}}^* s$ such that*

1. Qualitative Relationship: $s \equiv \mathbf{m}(u)$;
2. Quantitative Relationship: $|d|_{\beta_v^b} = |e|_e$;
3. Normal Forms: *if u is shuf-normal then $s, \mathbf{m}(u)$ are vsub-normal.*

Proof. First, by straightforward induction on $|d|_{\text{shuf}} \in \mathbb{N}$ using the projection via **m**-normal forms (Lemmas 14.1-2), one proves that there is $e_1: \mathbf{m}(t) \rightarrow_{\text{vsub}}^* \mathbf{m}(u)$ with $|e_1|_e = |d|_{\beta_v^b}$. By postponement of \equiv (Lemma 5.2), there is $e_2: \mathbf{m}(t) \rightarrow_{\text{vsub}}^* \mathbf{m}(u)$ with $|e_2|_e = |e_1|_e$. Clearly, $t \rightarrow_{\mathbf{m}}^* \mathbf{m}(t)$. It easy to check that $s \equiv r$ implies

$s \not\rightarrow_e r$ for all $s, r \in \Lambda_{\text{vsub}}$. Therefore, there exist $s \in \Lambda_{\text{vsub}}$ and $e: t \rightarrow_{\text{vsub}}^* s$ such that $s \equiv \mathfrak{m}(u)$ and $|e|_e = |e_2|_e = |d|_{\beta_v^b}$.

Finally, if moreover u is shuf-normal then, since normal forms are preserved by multiplicative projection (Lemma 15), $\mathfrak{m}(u)$ is vsub-normal, and hence so is s (Lemma 5.3, because $s \equiv \mathfrak{m}(u)$). \square

Corollary 17 (Termination Equivalence of λ_{vsub} and λ_{shuf}). *Let $t \in \Lambda$. There is a shuf-normalizing derivation d from t iff there is a vsub-normalizing derivation e from t . Moreover, $|d|_{\beta_v^b} = |e|_e$.*

See p. 16

Proof.

\Rightarrow : Let $d: t \rightarrow_{\text{shuf}}^* u$ be a shuf-normalizing derivation and $e: t \rightarrow_{\text{vsub}}^* s$ be its projection in λ_{vsub} with s vsub-normal, according to Thm. 12. Then e is a vsub-normalizing derivation from t .

\Leftarrow : By contradiction, suppose that there is a diverging shuf-derivation d from t in λ_{shuf} . Since \rightarrow_{σ^b} is strongly normalizing (Prop. 6.2), necessarily in d there are infinitely many β_v^b -steps. By Thm. 12, d projects to a vsub-derivation in λ_{vsub} that has as many e -steps as the β_v^b -steps in λ_{shuf} , absurd.

About the length, we have $|d|_{\beta_v^b} = |e|_e$ by Thm. 12.2. \square

Corollary 18 (Number of β_v^b -Steps is Invariant). *All shuf-normalizing derivations from $t \in \Lambda$ (if any) have the same number of β_v^b -steps.*

See p. 16

Proof. Let $d: t \rightarrow_{\text{shuf}}^* u$ and $d': t \rightarrow_{\text{shuf}}^* u'$ be shuf-normalizing. By confluence of $\rightarrow_{\text{shuf}}$ (Prop. 6.3), $u = u'$. According to Thm. 16, d and d' project, respectively, to two vsub-normalizing derivations $e: t \rightarrow_{\text{vsub}}^* s \in \Lambda_{\text{vsub}}$ and $e': t \rightarrow_{\text{vsub}}^* s' \in \Lambda_{\text{vsub}}$ such that $s \equiv \mathfrak{m}(u) \equiv s'$, $|e|_e = |d|_{\beta_v^b}$ and $|e'|_e = |d'|_{\beta_v^b}$. By Prop. 4.3, $|e|_e = |e'|_e$ and hence $|d|_{\beta_v^b} = |d'|_{\beta_v^b}$. \square

B.3 Proofs of Section 4 (Quantitative Equivalence of λ_{vsub} and λ_{vseq} , via λ_{vsub_k})

Proofs of Subsection 4.1 (Equivalence of λ_{vsub} and λ_{vsub_k}) We first give some more details on λ_{vsub_k} .

The kernel λ_{vsub_k} of λ_{vsub} is the sublanguage of λ_{vsub} defined by the following grammar of terms and values (by mutual induction), and evaluation contexts:

vsub _k -Values	$v ::= x \mid \lambda x.t$
vsub _k -Terms	$t, u, s ::= v \mid tv \mid t[x \leftarrow u]$
vsub _k -Evaluation Contexts	$E ::= \langle \cdot \rangle \mid Ev \mid E[x \leftarrow u] \mid t[x \leftarrow E]$

The top-level rewriting rules \mapsto_m and \mapsto_e are the same as in λ_{vsub} . Note that evaluation contexts of λ_{vsub_k} no longer include the case tE , because in λ_{vsub_k} such contexts cannot surround redexes, as E necessary is the empty context, that can only be filled in with a value, v , and values are normal forms. The set

of terms of λ_{vsub_k} is denoted by Λ_{vsub_k} . The restriction of $\rightarrow_{\text{vsub}}$ to Λ_{vsub_k} (i.e. the closure of $\mapsto_{\text{m}} \cup \mapsto_{\text{e}}$ under vsub_k -evaluation contexts) is denoted by $\rightarrow_{\text{vsub}_k}$. Note that vsub_k -terms are closed under substitution of vsub_k -values (i.e. if t is a vsub_k -term and v is a vsub_k -value, then $t\{x \leftarrow v\}$ is a vsub_k -term), therefore for every vsub_k -term t , if $t \rightarrow_{\text{vsub}} u$ then u is a vsub_k -term: so, $\rightarrow_{\text{vsub}_k}$ is a binary relation on Λ_{vsub_k} .

Lemma 44 (Substitution). *For any vsub-term t and any vsub-value v , one has $t\{x \leftarrow v\}^+ = t^+\{x \leftarrow v^+\}$.*

Proof. By induction on t . Cases:

- $t = x$: then, $t\{x \leftarrow v\} = v$ and $t^+ = x$, thus $t\{x \leftarrow v\}^+ = v^+ = t^+\{x \leftarrow v^+\}$.
- $t = y \neq x$: then, $t\{x \leftarrow v\} = y$ and $t^+ = y$, hence $t\{x \leftarrow v\}^+ = y = t^+\{x \leftarrow v^+\}$.
- $t = \lambda y.s$: we can suppose without loss of generality that $y \notin \text{fv}(v) \cup \{x\}$, whence $y \notin \text{fv}(v^+)$. By i.h., $s\{x \leftarrow v\}^+ = s^+\{x \leftarrow v^+\}$. So, $t\{x \leftarrow v\}^+ = \lambda y.s\{x \leftarrow v\}^+ = \lambda y.s^+\{x \leftarrow v^+\} = t^+\{x \leftarrow v^+\}$.
- $t = sr$: then, $t^+ = (s^+y)[y \leftarrow s^+]$ with $y \notin \text{fv}(s)$, and we can suppose without loss of generality that $y \notin \text{fv}(v) \cup \{x\}$. By i.h., $s\{x \leftarrow v\}^+ = s^+\{x \leftarrow v^+\}$ and $r\{x \leftarrow v\}^+ = r^+\{x \leftarrow v^+\}$, hence $t\{x \leftarrow v\}^+ = (s\{x \leftarrow v\}^+y)[y \leftarrow r\{x \leftarrow v\}^+] = (s^+\{x \leftarrow v^+\}y)[y \leftarrow r^+\{x \leftarrow v^+\}] = t^+\{x \leftarrow v^+\}$, since $y \neq x$.
- $t = s[y \leftarrow r]$: we can suppose without loss of generality that $y \notin \text{fv}(v) \cup \{x\}$. By i.h., $s\{x \leftarrow v\}^+ = s^+\{x \leftarrow v^+\}$ and $r\{x \leftarrow v\}^+ = r^+\{x \leftarrow v^+\}$, so $t\{x \leftarrow v\}^+ = s\{x \leftarrow v\}^+[y \leftarrow r\{x \leftarrow v\}^+] = s^+\{x \leftarrow v^+\}[y \leftarrow r^+\{x \leftarrow v^+\}] = t^+\{x \leftarrow v^+\}$. \square

See p. 17

Lemma 19 (Simulation). *Let $t, u \in \Lambda_{\text{vsub}}$.*

1. Multiplicative: *if $t \rightarrow_{\text{m}} u$ then $t^+ \rightarrow_{\text{m}} u^+$ or $t^+ \rightarrow_{\text{m}} \rightarrow_{\text{e}_y} u^+ \equiv u^+$;*
2. Exponential Abstractions & Variables: *if $t \rightarrow_{\text{e}_\lambda} u$ then $t^+ \rightarrow_{\text{e}_\lambda} u^+$, and if $t \rightarrow_{\text{e}_y} u$ then $t^+ \rightarrow_{\text{e}_y} u^+$.*
3. Structural Equivalence: *$t \equiv u$ implies $t^+ \equiv u^+$.*

Proof. Define L^+ by $\langle \cdot \rangle^+ := \langle \cdot \rangle$ and $L[x \leftarrow r]^+ := L^+[x \leftarrow r^+]$. Then note that if $t = L\langle s \rangle$ we have $t^+ = L^+\langle s^+ \rangle$. By induction on the evaluation context in which the step takes place. The base cases:

1. *Multiplicative, i.e. $t = L\langle \lambda x.s \rangle r \rightarrow_{\text{m}} L\langle s[x \leftarrow r] \rangle = u$.* Then

$$\begin{aligned}
 t^+ &= (L\langle \lambda x.s \rangle r)^+ \\
 &= (L^+\langle \lambda x.s^+ \rangle y)[y \leftarrow r^+] \\
 &\rightarrow_{\text{m}} L^+\langle s^+[x \leftarrow y] \rangle[y \leftarrow r^+] \\
 &\rightarrow_{\text{e}_y} L^+\langle s^+\{x \leftarrow y\} \rangle[y \leftarrow r^+] \\
 &\equiv L^+\langle s^+\{x \leftarrow y\}[y \leftarrow r^+] \rangle \\
 &=_{\alpha} L^+\langle s^+[x \leftarrow r^+] \rangle \\
 &= L\langle s[x \leftarrow r] \rangle^+ = u^+
 \end{aligned}$$

2. *Exponential Abstractions & Variables, i.e. $t = s[x \leftarrow L\langle v \rangle] \rightarrow_{\text{e}} L\langle s\{x \leftarrow v\} \rangle = u$.*

$$\begin{aligned}
t^+ &= s[x \leftarrow L\langle v \rangle]^+ \\
&= s^+[x \leftarrow L^+\langle v^+ \rangle] \\
&\rightarrow_e L^+\langle s^+\{x \leftarrow v^+\} \rangle \\
&\stackrel{=_{L.44}}{=} L^+\langle s\{x \leftarrow v\}^+ \rangle \\
&= L\langle s\{x \leftarrow v\} \rangle^+ = u^+
\end{aligned}$$

Note that the translation \cdot^+ maps \rightarrow_{e_λ} steps on \rightarrow_{e_λ} steps and \rightarrow_{e_y} steps to \rightarrow_{e_y} steps because it maps variables to variables and abstractions to abstractions.

3. *Structural Equivalence*: it is enough to prove that the statement holds for the axioms of \equiv , i.e. if $t \equiv_r u$ for some $r \in \{\text{com}, @_l, @_r, [\cdot]\}$, then $t^+ \equiv u^+$. Cases (recall that vsub_k is a sublanguage of vsub):

- $t[y \leftarrow s][x \leftarrow u] \equiv_{\text{com}} t[x \leftarrow u][y \leftarrow s]$ with $y \notin \text{fv}(u)$ and $x \notin \text{fv}(s)$: then, $t[y \leftarrow s][x \leftarrow u]^+ = t^+[y \leftarrow s^+][x \leftarrow u^+] \equiv_{\text{com}} t^+[x \leftarrow u^+][y \leftarrow s^+] = t[x \leftarrow u][y \leftarrow s]^+$, since $y \notin \text{fv}(u^+)$ and $x \notin \text{fv}(s^+)$.
- $t s[x \leftarrow u] \equiv_{@_r} (ts)[x \leftarrow u]$ with $x \notin \text{fv}(t)$: then, $x \notin \text{fv}(t^+)$ and $(ts)^+ = (t^+y)[y \leftarrow s^+]$ with $y \notin \text{fv}(t) = \text{fv}(t^+)$, and we can suppose without loss of generality that $y \neq x$. So, $(ts[x \leftarrow u])^+ = (t^+y)[y \leftarrow s^+][x \leftarrow u^+] \equiv_{[\cdot]} (t^+y)[y \leftarrow s^+][x \leftarrow u^+] = (ts)[x \leftarrow u]^+$.
- $t[x \leftarrow u]s \equiv_{@_l} (ts)[x \leftarrow u]$ with $x \notin \text{fv}(s)$: then, $x \notin \text{fv}(s^+)$ and $(ts)^+ = (t^+y)[y \leftarrow s^+]$ with $y \notin \text{fv}(t) = \text{fv}(t^+)$. We can suppose without loss of generality that $y \notin \text{fv}(u) \cup \{x\}$, hence $t^+[x \leftarrow u^+]y \equiv_{@_l} (t^+y)[x \leftarrow u^+]$. Therefore, $(t[x \leftarrow u]s)^+ = (t^+[x \leftarrow u^+]y)[y \leftarrow s^+] \equiv (t^+y)[x \leftarrow u^+][y \leftarrow s^+] \equiv_{[\cdot]} (t^+y)[y \leftarrow s^+][x \leftarrow u^+] = (ts)[x \leftarrow u]^+$.
- $t[x \leftarrow u][y \leftarrow s] \equiv_{[\cdot]} t[x \leftarrow u][y \leftarrow s]$ with $y \notin \text{fv}(t)$: then, $t[x \leftarrow u][y \leftarrow s]^+ = t^+[x \leftarrow u^+][y \leftarrow s^+] \equiv_{[\cdot]} t^+[x \leftarrow u^+][y \leftarrow s^+] = t[x \leftarrow u][y \leftarrow s]^+$, since $y \notin \text{fv}(t^+)$.

The inductive cases simply follow from the *i.h.*, note indeed that evaluation contexts are translated to evaluation contexts. \square

Definition 45. Let t be a vsub -term: t is harmless if for every subterm of the form $u[x \leftarrow L\langle v \rangle]$ occurring in t one has $u = sv'$ with $x \notin \text{fv}(s)$ (where v and v' are vsub -values).

Lemma 46 (Harmless preservations). Let $t \in \Lambda_{\text{vsub}}$.

1. Suppose t is harmless and $t \rightarrow_e u$. Then, u is harmless. If, moreover, t is \mathbf{m} -normal, then u is \mathbf{m} -normal.
2. If t is vsub -normal, then t^+ is harmless and \mathbf{m} -normal.

Proof.

1. From $t \rightarrow_e u$ it follows that u is obtained from t by replacing a subterm of the form $t'[x \leftarrow L\langle v \rangle]$ with $L\langle t'\{x \leftarrow v\} \rangle$, where all the variables bound by the substitution context L are not free in t' . Since t is harmless, then $t' = sv'$ with $x \notin \text{fv}(s)$ and hence $L\langle t'\{x \leftarrow v\} \rangle = L\langle s(v'\{x \leftarrow v\}) \rangle$ where $v'\{x \leftarrow v\}$ is a vsub -value, since vsub -values are closed by substitution. Moreover, all the variables

binded by the substitution context L are not free in s . So, $L\langle t'\{x \leftarrow v\} \rangle$ is not an objection to the harmless property. Therefore, u is harmless.

If u is not \mathbf{m} -normal then it has a subterm of the form $L\langle \lambda x.s \rangle r$, but t is harmless and hence the step $t \rightarrow_e u$ can only substitute a value for a variable in argument position of an application: this means that t has a subterm of the form $L\langle \lambda x.s \rangle r'$ and hence t is not \mathbf{m} -normal.

2. The translation $(\cdot)^+$ essentially creates new ES from applications, but does not create new abstractions or new applications, hence if t^+ contained a subterm of the form $L\langle \lambda x.u \rangle s$ then t would contain a subterm of the form $L'\langle \lambda x.u' \rangle s'$: but t is \mathbf{m} -normal and therefore t^+ is \mathbf{m} -normal. Moreover, since t is \mathbf{e} -normal, the only subterms of t^+ of the form $u[x \leftarrow L\langle v \rangle]$ are created by the translation $(\cdot)^+$ (i.e. they are not in t), and the ES introduced by the translation $(\cdot)^+$ fulfill the harmless condition. Therefore t^+ is harmless. \square

See p. 17

Theorem 20 (Quantitative Simulation of λ_{vsub} in λ_{vsub_k}). *Let $t, u \in \Lambda_{\text{vsub}}$. If $d: t \rightarrow_{\text{vsub}}^* u$ then there are $s \in \Lambda_{\text{vsub}_k}$ and $e: t^+ \rightarrow_{\text{vsub}_k}^* s$ such that*

1. Qualitative Relationship: $s \equiv u^+$;
2. Quantitative Relationship:
 1. Multiplicative Steps: $|e|_{\mathbf{m}} = |d|_{\mathbf{m}}$;
 2. Exponential Steps: $|e|_{\mathbf{e}_\lambda} = |d|_{\mathbf{e}_\lambda}$ and $|e|_{\mathbf{e}_y} = |d|_{\mathbf{e}_y} + |d|_{\mathbf{m}}$;
3. Normal Form: if u is vsub -normal then s is \mathbf{m} -normal and $\mathbf{e}(s)$ is vsub_k -normal.

Proof.

- 1-2. By induction on $|d|$ using Lemma 19.1 and Lemma 19.2 plus the postponement of \equiv (Lemma 5.1).
3. By Lemma 46.2, u^+ is \mathbf{m} -normal and harmless, thus also $\mathbf{e}(u^+)$ is \mathbf{m} -normal according to Lemma 46.1. Therefore, $\mathbf{e}(u^+)$ is vsub -normal. By the properties of strong bisimulation (Lemma 5), $s \equiv u^+$ implies that $\mathbf{e}(s) \equiv \mathbf{e}(u^+)$ and hence $\mathbf{e}(s)$ is vsub -normal. \square

See p. 17

Corollary 21 (Linear Termination Equivalence of λ_{vsub} and λ_{vsub_k}). *Let $t \in \Lambda_{\text{vsub}}$. There exists a vsub -normalizing derivation d from t iff there exists a vsub_k -normalizing derivation f from t^+ . Moreover, $|d|_{\mathbf{m}} = |f|_{\mathbf{m}}$.*

Proof.

- \Rightarrow : Let $d: t \rightarrow_{\text{vsub}}^* u$ be a vsub -normalizing derivation and $e: t^+ \rightarrow_{\text{vsub}_k}^* s$ be its projection in λ_{vsub_k} , according to Thm. 20. By Thm. 20.3, the derivation f obtained by extending e with a normalization with respect to \rightarrow_e (that always terminate) is a vsub_k -normalizing derivation from t^+ .
- \Leftarrow : By contradiction, suppose that there is a diverging vsub -derivation from t in λ_{vsub} . By Thm. 20 it projects to a vsub_k -derivation from t^+ in λ_{vsub_k} that is at least as long as the one in λ_{vsub} , absurd.

About lengths, consider the normalizing derivations d , e , and f discussed in the proof of the \Rightarrow direction. We have to show that $|d|_{\mathbf{m}} = |f|_{\mathbf{m}}$. By Thm. 20.3, $|d|_{\mathbf{m}} = |e|_{\mathbf{m}}$. Since f extends e only with exponential steps, we obtain $|f|_{\mathbf{m}} = |e|_{\mathbf{m}}$. \square

Proof of Subsection 4.2 (Equivalence of λ_{vsub_k} and λ_{vseq})

Lemma 47 (Translation and Substitution Commute). *Let v, v' be values and t be a term of vsub_k .*

1. Values: $(v'\{x \leftarrow v\})^\bullet = v'^\bullet\{x \leftarrow v^\bullet\}$;
2. Terms: $\underline{t\{x \leftarrow v\}} = \underline{t}\{x \leftarrow v^\bullet\}$.

Proof.

1. Cases:
 - Variable, i.e. $t = x$. Then $(x\{x \leftarrow v\})^\bullet = v^\bullet = x\{x \leftarrow v^\bullet\} = x^\bullet\{x \leftarrow v^\bullet\}$.
 - Abstraction, i.e. $t = \lambda y.u$.

$$\begin{aligned} ((\lambda y.u)\{x \leftarrow v\})^\bullet &= (\lambda y.u\{x \leftarrow v\})^\bullet \\ &= \lambda y.u\{x \leftarrow v\} \\ &=_{P.2} \lambda y.\underline{u}\{x \leftarrow v^\bullet\} \\ &= (\lambda y.\underline{u})\{x \leftarrow v^\bullet\} = (\lambda y.u)^\bullet\{x \leftarrow v^\bullet\} \end{aligned}$$

2. By induction on t . Cases:
 - Value, i.e. $t = v'$. Note that $v'\{x \leftarrow v\}$ is a value. Then $\underline{v'\{x \leftarrow v\}} = \langle (v'\{x \leftarrow v\})^\bullet | \epsilon \rangle =_{P.1} \langle v'^\bullet\{x \leftarrow v^\bullet\} | \epsilon \rangle = \langle v'^\bullet | \epsilon \rangle\{x \leftarrow v^\bullet\} = \underline{v'}\{x \leftarrow v^\bullet\}$.
 - Application, i.e. $t = uv'$.

$$\begin{aligned} \underline{uv'\{x \leftarrow v\}} &= \underline{u\{x \leftarrow v\}v'\{x \leftarrow v\}} \\ &= \underline{u\{x \leftarrow v\}} @ (v'\{x \leftarrow v\}^\bullet \cdot \epsilon) \\ &=_{i.h.} \underline{u}\{x \leftarrow v^\bullet\} @ (v'\{x \leftarrow v\}^\bullet \cdot \epsilon) \\ &=_{P.1} \underline{u}\{x \leftarrow v^\bullet\} @ (v'^\bullet\{x \leftarrow v^\bullet\} \cdot \epsilon) \\ &= \underline{u} @ (v'^\bullet \cdot \epsilon)\{x \leftarrow v^\bullet\} = \underline{uv'}\{x \leftarrow v^\bullet\} \end{aligned}$$

- Substitution, i.e. $t = u[y \leftarrow s]$.

$$\begin{aligned} \underline{u[y \leftarrow s]\{x \leftarrow v\}} &= \underline{u\{x \leftarrow v\}[y \leftarrow s\{x \leftarrow v\}]} \\ &= \underline{s\{x \leftarrow v\}} @ \tilde{\mu}y.u\{x \leftarrow v\} \\ &=_{i.h.} \underline{s}\{x \leftarrow v^\bullet\} @ \tilde{\mu}y.\underline{u}\{x \leftarrow v^\bullet\} \\ &= (\underline{s} @ \tilde{\mu}y.\underline{u})\{x \leftarrow v^\bullet\} = \underline{u[y \leftarrow s]}\{x \leftarrow v^\bullet\} \end{aligned}$$

□

Lemma 48. *Let t be a vsub_k -term. Then there exist a command evaluation context C and an environment evaluation context D such that $\underline{t} = C\langle D\langle \epsilon \rangle \rangle$.*

Proof. By induction on t . Cases:

1. Variable, i.e. $t = x$. Trivial just take $C := \langle \cdot \rangle$ and $D := \langle x | \langle \cdot \rangle \rangle$.
2. Abstraction, i.e. $t = \lambda x.u$. Trivial just take $C := \langle \cdot \rangle$ and $D := \langle \lambda x.\underline{u} | \langle \cdot \rangle \rangle$.
3. Application, i.e. $t = uv$.

$$\begin{aligned} \underline{uv} &= \underline{u} @ (v^\bullet \cdot \epsilon) \\ &=_{i.h.} C'\langle D'\langle \epsilon \rangle \rangle @ (v^\bullet \cdot \epsilon) \\ &= C'\langle D'\langle v^\bullet \cdot \epsilon \rangle \rangle \end{aligned}$$

The statement holds with respect to $C := C'$ and $D := D'\langle x \cdot \langle \cdot \rangle \rangle$.

4. *Substitution*, i.e. $t = u[x \leftarrow s]$.

$$\begin{aligned} \underline{u[x \leftarrow s]} &= \underline{s @ \tilde{\mu}x.u} \\ &=_{i.h.} C' \langle D' \langle \epsilon \rangle \rangle @ \tilde{\mu}x.\underline{u} \\ &= C' \langle D' \langle \tilde{\mu}x.\underline{u} \rangle \rangle \\ &=_{i.h.} C' \langle D' \langle \tilde{\mu}x.C'' \langle D'' \langle \epsilon \rangle \rangle \rangle \end{aligned}$$

The statement holds with respect to $C := C' \langle D' \langle \tilde{\mu}x.C'' \rangle \rangle$ and $D := D''$. \square

Lemma 49. *Let L be a substitution context of **vsub**. There exists a command evaluation context C such that $\underline{L \langle t \rangle} = C \langle \underline{t} \rangle$ for any **vsub**_k-term t . Moreover, $\text{fv}(C) = \text{fv}(L)$ and C and L capture the same variables of t .*

Proof. By induction on L . Cases:

1. *Empty Context*, i.e. $L = \langle \cdot \rangle$. Just take $C := \langle \cdot \rangle$.
2. *Non-Empty Context*, i.e. $L = L'[x \leftarrow u]$. Then

$$\begin{aligned} \underline{L' \langle t \rangle [x \leftarrow u]} &= \underline{u @ \tilde{\mu}x.L' \langle t \rangle} \\ &=_{i.h.} \underline{u @ \tilde{\mu}x.C' \langle \underline{t} \rangle} \\ &=_{L.48} C'' \langle D \langle \epsilon \rangle \rangle @ \tilde{\mu}x.C' \langle \underline{t} \rangle \\ &= C'' \langle D \langle \tilde{\mu}x.C' \langle \underline{t} \rangle \rangle \rangle \end{aligned}$$

The statement holds with respect to $C := C'' \langle D \langle \tilde{\mu}x.C' \rangle \rangle$. The *moreover* part follows from the *moreover* part of Lemma 48 and the *i.h.* \square

See p. 18

Lemma 22 (Simulation of $\rightarrow_{\text{vsub}_k}$ by $\rightarrow_{\text{vseq}}$). *Let $t, u \in \Lambda_{\text{vsub}_k}$.*

1. *Multiplicative*: if $t \rightarrow_m u$ then $\underline{t} \rightarrow_{\bar{\lambda}} \underline{u}$.
2. *Exponential*: if $t \rightarrow_e u$ then $\underline{t} \rightarrow_{\bar{\mu}} \underline{u}$.

Proof. Both points are proved by induction on the evaluation context E in which the step takes place. Cases:

- *Root case*, i.e. $E = \langle \cdot \rangle$.
 1. *Multiplicative Step*: $t = L \langle \lambda x.s \rangle v \mapsto_m L \langle s[x \leftarrow v] \rangle = u$.

$$\begin{aligned} \underline{L \langle \lambda x.s \rangle v} &= \underline{L \langle \lambda x.s \rangle @ (v^\bullet \cdot \epsilon)} \\ &=_{L.49} C' \langle \underline{\lambda x.s} \rangle @ (v^\bullet \cdot \epsilon) \\ &= C' \langle \langle \lambda x.\underline{s} | \epsilon \rangle \rangle @ (v^\bullet \cdot \epsilon) \\ &= C' \langle \langle \lambda x.\underline{s} | \epsilon \rangle @ (v^\bullet \cdot \epsilon) \rangle \\ &= C' \langle \langle \lambda x.\underline{s} | v^\bullet \cdot \epsilon \rangle \rangle \\ &\rightarrow_{\bar{\lambda}} C' \langle \langle v^\bullet | \tilde{\mu}x.\underline{s} @ \epsilon \rangle \rangle \\ &= C' \langle \langle v^\bullet | \tilde{\mu}x.\underline{s} \rangle \rangle \\ &= C' \langle \underline{s[x \leftarrow v]} \rangle =_{L.49} \underline{L \langle s[x \leftarrow v] \rangle} \end{aligned}$$

2. *Exponential Step*: $t = s[x \leftarrow L \langle v \rangle] \mapsto_e L \langle s[x \leftarrow v] \rangle = u$.

$$\begin{aligned}
\underline{s[x \leftarrow L\langle v \rangle]} &= \underline{L\langle v \rangle @ \tilde{\mu}x.\underline{s}} \\
&=_{L.49} \underline{C\langle \underline{v} \rangle @ \tilde{\mu}x.\underline{s}} \\
&= C\langle \langle v^\bullet | \epsilon \rangle \rangle @ \tilde{\mu}x.\underline{s} \\
&= C\langle \langle v^\bullet | \tilde{\mu}x.\underline{s} \rangle \rangle \\
&\rightarrow_{\tilde{\mu}} C\langle \underline{s}\{x \leftarrow v^\bullet\} \rangle \\
&=_{L.47} \underline{C\langle \underline{s}\{x \leftarrow v\} \rangle} =_{L.49} \underline{L\langle s\{x \leftarrow v\} \rangle}
\end{aligned}$$

- *Inductive Cases:* for each case the two points differs only in the kind of the rewriting step, so we treat them compactly, by referring to $\rightarrow_{\text{vsub}_k}$ and $\rightarrow_{\text{vseq}}$
 - *Left Application,* i.e. $t = sv \rightarrow_{\text{vsub}_k} rv = u$ with $s \rightarrow_{\text{vsub}_k} r$. By i.h., $\underline{s} \rightarrow_{\text{vseq}} \underline{r}$. And by Lemma 39.2, $\underline{sv} = \underline{s} @ (\underline{v} \cdot \epsilon) \rightarrow_{\text{vseq}} \underline{r} @ (\underline{v} \cdot \epsilon) = \underline{rv}$.
 - *Left of a Substitution:* i.e. $t = s[x \leftarrow q] \rightarrow_{\text{vsub}_k} r[x \leftarrow q] = u$ with $s \rightarrow_{\text{vsub}_k} r$. By i.h., $\underline{s} \rightarrow_{\text{vseq}} \underline{r}$. By Lemma 38, $\underline{s[x \leftarrow q]} = \underline{q} @ \tilde{\mu}x.\underline{s} \rightarrow_{\text{vseq}} \underline{q} @ \tilde{\mu}x.\underline{r} = \underline{s[x \leftarrow r]}$.
 - *Inside a Substitution:* i.e. $t = s[x \leftarrow q] \rightarrow_{\text{vsub}_k} s[x \leftarrow r] = u$ with $q \rightarrow_{\text{vsub}_k} r$. By i.h., $\underline{q} \rightarrow_{\text{vseq}} \underline{r}$. And by Lemma 39.2, $\underline{s[x \leftarrow q]} = \underline{q} @ \tilde{\mu}x.\underline{s} \rightarrow_{\text{vseq}} \underline{r} @ \tilde{\mu}x.\underline{s} = \underline{s[x \leftarrow r]}$. \square

Theorem 23 (Quantitative Simulation of λ_{vsub_k} in λ_{vseq}). *Let $t, u \in \Lambda_{\text{vsub}_k}$. If $d: t \rightarrow_{\text{vsub}_k}^* u$ then there is $e: \underline{t} \rightarrow_{\text{vseq}}^* \underline{u}$ such that $|e|_{\text{vseq}}$ denotes the length of e*

See p. 18

1. Multiplicative Steps: $|d|_m = |e|_{\bar{\lambda}}$ (the number $\bar{\lambda}$ -steps in e);
2. Exponential Steps: $|d|_e = |e|_{\tilde{\mu}}$ (the number $\tilde{\mu}$ -steps in e), so $|d|_{\text{vsub}_k} = |e|_{\text{vseq}}$;
3. Normal Form: if u is vsub_k -normal then \underline{u} is vseq -normal.

Proof. The existence of e and the first two points are immediate consequences of Lemma 22. We prove Point 3 by proving that the translation of a clean normal form t of λ_{vsub_k} is normal. Cases of u :

- *Value:* then clearly $\underline{u} = \langle u^\bullet | \epsilon \rangle$ is normal.
- *Compound Inert Term:* then u has the form $u = xv_1 \dots v_k$. A straightforward induction on k shows that it translates to $\langle x | v_1^\bullet \dots v_k^\bullet \cdot \epsilon \rangle$, that is normal.
- *Substitution:* then u has the form $u = s[x \leftarrow i]$ where s is a clean normal form and i is a compound inert term. If $i = yv_1 \dots v_k$ then $\underline{i} = \langle y | v_1^\bullet \dots v_k^\bullet \cdot \epsilon \rangle$ and $\underline{s[x \leftarrow i]} = \langle y | v_1^\bullet \dots v_k^\bullet \cdot \epsilon \rangle @ \tilde{\mu}x.\underline{s} = \langle y | v_1^\bullet \dots v_k^\bullet \cdot \tilde{\mu}x.\underline{s} \rangle$, that is normal because by i.h. \underline{s} is normal. \square

Corollary 24 (Linear Termination Equivalence of λ_{vsub_k} and λ_{vseq}). *Let $t \in \Lambda_{\text{vsub}_k}$. There is a vsub_k -normalizing derivation d from t iff there is a vseq -normalizing derivation e from \underline{t} . Moreover, $|d|_{\text{vsub}_k} = |e|_{\text{vseq}}$, $|d|_e = |e|_{\tilde{\mu}}$ and $|d|_m = |e|_{\bar{\lambda}}$.*

See p. 18

Proof. \Rightarrow : Let $d: t \rightarrow_{\text{vsub}_k}^* u$ be a vsub_k -normalizing derivation and $e: \underline{t} \rightarrow_{\text{vseq}}^* \underline{u}$ be its projection in λ_{vseq} , according to Thm. 23. Then e is a vsub_k -normalizing derivation from \underline{t} , since the vsub_k -normality of u implies the vseq -normality of \underline{u} by Thm. 23.3.

\Leftarrow : By contradiction, suppose that there is a diverging vsub_k -derivation from t in λ_{vsub_k} . By Thm. 23 it projects to a vseq -derivation from \underline{t} in λ_{vseq} that is at least as long as the one in λ_{vsub_k} , which is absurd since \underline{t} is vseq -normalizable and all vseq -normalizing derivations from \underline{t} have the same length by Prop. 7.3. The result about lengths follows immediately from Thm. 23.1-2. \square

Structural equivalence for λ_{vseq} .

Remark 50. Every environment evaluation context can be uniquely written as $D = \langle v | v_1 \dots v_n \cdot e \rangle$ where either $e = \epsilon$ or $e = \tilde{\mu}x.c$.

Lemma 51. *Let t and u be vsub_k -terms.*

1. *If $t \equiv_{[\cdot], @1} t'$ then $\underline{t} = \underline{t}'$.*
2. *If $t \equiv_{\text{com}} t'$ then $\underline{t} \simeq_{\tilde{\mu}\tilde{\mu}} \underline{t}'$.*

Proof. 1. If $t \equiv_{[\cdot]} t'$, then $t = s[x \leftarrow r[y \leftarrow u]]$ and $t' = s[x \leftarrow r][y \leftarrow u]$. So, just apply the translation and Lemma 39.1 (setting $C = \underline{u} @ \tilde{\mu}y. \langle \cdot \rangle$, $c = \underline{r}$ and $e = \tilde{\mu}x.\underline{s}$):

$$\underline{t} = \underline{s[x \leftarrow r[y \leftarrow u]]} = (\underline{u} @ \tilde{\mu}y. \underline{r}) @ \tilde{\mu}x. \underline{s} = \underline{u} @ \tilde{\mu}y. (\underline{r} @ \tilde{\mu}x. \underline{s}) = \underline{s[x \leftarrow r][y \leftarrow u]} = \underline{t}'.$$

If $t \equiv_{@1} t'$, then $t = s[x \leftarrow u]v$ and $t' = (sv)[x \leftarrow u]$. So, just apply the translation and Lemma 39.1 (setting $C = \underline{u} @ \tilde{\mu}x. \langle \cdot \rangle$, $c = \underline{s}$ and $e = v \bullet \epsilon$):

$$\underline{t} = \underline{s[x \leftarrow u]v} = (\underline{u} @ \tilde{\mu}x. \underline{s}) @ (v \bullet \epsilon) = \underline{u} @ \tilde{\mu}x. (\underline{s} @ (v \bullet \epsilon)) = \underline{(sv)[x \leftarrow u]} = \underline{t}'.$$

2. As $t \equiv_{\text{com}} t'$, then $t = s[y \leftarrow r][x \leftarrow u]$ and $t' = s[x \leftarrow u][y \leftarrow r]$ with $x \notin \text{fv}(s)$ and $y \notin \text{fv}(r)$. So, just apply the translation and the definition of $\simeq_{\tilde{\mu}\tilde{\mu}}$, the only axiom generating \simeq (setting $D = \underline{u} @ \langle \cdot \rangle$ and $D' = \underline{r} @ \langle \cdot \rangle$):

$$\underline{t} = \underline{s[y \leftarrow r][x \leftarrow u]} = \underline{u @ \tilde{\mu}x. (\underline{r} @ \tilde{\mu}y. \underline{s})} \simeq_{\tilde{\mu}\tilde{\mu}} \underline{r @ \tilde{\mu}y. (\underline{u} @ \tilde{\mu}x. \underline{s})} = \underline{s[x \leftarrow u][y \leftarrow r]} = \underline{t}'.$$

□

Proposition 52 (Simulation of \equiv by \simeq). *Let t and t' be vsub_k -terms. If $t \equiv t'$ then $\underline{t} \simeq \underline{t}'$.*

Proof. First, observe that there are no $u, u' \in \Lambda_{\text{vsub}_k}$ such that $u \equiv_{@r} u'$: indeed, $u \equiv_{@r} u'$ implies that $u = sr[x \leftarrow q]$ and $r[x \leftarrow q]$ is not a value, therefore $u \notin \Lambda_{\text{vsub}_k}$.

Let \equiv' the closure of $\equiv_{@1} \cup \equiv_{\text{com}} \cup \equiv_{[\cdot]}$ under evaluation contexts of λ_{vsub_k} . As \equiv on Λ_{vsub_k} is just the reflexive-transitive and symmetric closure of \equiv (and \simeq is an equivalence relation), in order to prove Prop. 52 it is enough to prove that the following statement (*): for every $t, t' \in \Lambda_{\text{vsub}_k}$, if $t \equiv' t'$ then $\underline{t} \simeq \underline{t}'$. The proof of (*) is by induction on the definition $t \equiv' t'$.

The base cases (*i.e.* when $t \equiv_{@1} t'$ or $t \equiv_{\text{com}} t'$ or $t \equiv_{[\cdot]} t'$) are already proved in Lemma 51. Concerning the inductive cases, we have:

- *Application Left, i.e.* $t := uv \equiv u'v =: t'$ with $u \equiv u'$: by *i.h.*, $\underline{u} \simeq \underline{u}'$; so, $\underline{t} = \underline{u} @ (v \bullet \epsilon) \simeq \underline{u}' @ (v \bullet \epsilon) = \underline{t}'$;
- *Left of a Substitution, i.e.* $t := u[x \leftarrow s] \equiv u'[x \leftarrow s] =: t'$ with $u \equiv u'$: by *i.h.*, $\underline{u} \simeq \underline{u}'$; so, $\underline{t} = \underline{s} @ \tilde{\mu}x. \underline{u} \simeq \underline{s} @ \tilde{\mu}x. \underline{u}' = \underline{t}'$;
- *Inside a Substitution, i.e.* $t := s[x \leftarrow u] \equiv s[x \leftarrow u'] =: t'$ with $u \equiv u'$: by *i.h.*, $\underline{u} \simeq \underline{u}'$; thus, $\underline{t} = \underline{u} @ \tilde{\mu}x. \underline{s} \simeq \underline{u}' @ \tilde{\mu}x. \underline{s} = \underline{t}'$. □

Proposition 53 (Basic Properties of Structural Equivalence \simeq). *Let c_0, c_1 be commands and $r \in \{\tilde{\lambda}, \tilde{\mu}, \text{vseq}\}$.*

1. Strong Bisimulation of \simeq wrt $\rightarrow_{\text{vseq}}$: if $c_0 \simeq c_1$ and $c_1 \rightarrow_r c_2$ then there exists a command c_3 such that $c_0 \rightarrow_r c_3 \simeq c_2$.
2. Postponement of \simeq wrt $\rightarrow_{\text{vseq}}$: if $d: c_0 \rightarrow_{\text{vseq}}^* c_1$ then there are $c_2 \simeq c_1$ and $e: c_0 \rightarrow_{\text{vseq}}^* c_2$ such that $|d|_{\text{vseq}} = |e|_{\text{vseq}}$, $|d|_{\tilde{\mu}} = |e|_{\tilde{\mu}}$ and $|d|_{\bar{\lambda}} = |e|_{\bar{\lambda}}$.
3. Normal Forms: if $t \simeq u$ then t is r -normal iff u is r -normal.
4. Strong confluence: $\rightarrow_{\text{vseq}} \simeq$ is strongly confluent.

Proof. 1. It is enough to prove the following statement (*): if $c_0 \simeq' c_1$ and $c_1 \rightarrow_r c_2$ then there exists a command c_3 such that $c_0 \rightarrow_r c_3 \simeq' c_2$, where \simeq' is the reflexive closure under command evaluation contexts of $\simeq_{\tilde{\mu}\tilde{\mu}}$, the unique axiom generating the equivalence \simeq . Indeed \simeq' is reflexive and symmetric, therefore \simeq is just the transitive closure of \simeq' , so the proof of Prop. 53.1 follows immediately.

The proof of (*) is by induction on the definition of \simeq' .

In the inductive cases the proof follows immediately from the *i.h.*, since \simeq' and \rightarrow_r are closed under the same contexts.

Concerning the base cases, according to Remark 50, we have

$$\begin{aligned} c_0 &:= \langle v | v_1 \dots v_n \cdot \tilde{\mu}x. \langle v' | v'_1 \dots v'_{n'} \cdot \tilde{\mu}y.c \rangle \rangle \\ &\simeq_{\tilde{\mu}\tilde{\mu}} \langle v' | v'_1 \dots v'_{n'} \cdot \tilde{\mu}y. \langle v | v_1 \dots v_n \cdot \tilde{\mu}x.c \rangle \rangle =: c_1 \end{aligned}$$

where $x \notin \text{fv}(v') \cup \bigcup_{i'=1}^{n'} \text{fv}(v'_{i'})$ and $y \notin \text{fv}(v) \cup \bigcup_{i=1}^n \text{fv}(v_i)$. Thus there are only four cases:

(a) *Internal $\bar{\lambda}$ -step*, i.e. $v = \lambda z.c'$, $n > 0$ and

$$c_1 \rightarrow_{\bar{\lambda}} \langle v' | v'_1 \dots v'_{n'} \cdot \tilde{\mu}y. \langle v_1 | (\tilde{\mu}z.c') @ (v_2 \dots v_n \cdot \tilde{\mu}x.c) \rangle \rangle = c_2$$

then, $c_0 \rightarrow_{\bar{\lambda}} \langle v_1 | (\tilde{\mu}z.c') @ (v_2 \dots v_n \cdot \tilde{\mu}x. \langle v' | v'_1 \dots v'_{n'} \cdot \tilde{\mu}y.c \rangle) \rangle \simeq' c_2$, where the last equivalence holds by applying the axiom $\simeq_{\tilde{\mu}\tilde{\mu}}$ with the environment evaluation contexts $D = \langle v_1 | (\tilde{\mu}z.c') @ v_2 \dots v_n \cdot \langle \cdot \rangle \rangle$ and $D' = \langle v' | v'_1 \dots v'_{n'} \cdot \langle \cdot \rangle \rangle$.

(b) *External $\bar{\lambda}$ -step*, i.e. $v' = \lambda z.c'$, $n' > 0$ and

$$c_1 \rightarrow_{\bar{\lambda}} \langle v'_1 | (\tilde{\mu}z.c') @ (v'_2 \dots v'_{n'} \cdot \tilde{\mu}y. \langle v | v_1 \dots v_n \cdot \tilde{\mu}x.c \rangle) \rangle = c_2$$

then, $c_0 \rightarrow_{\bar{\lambda}} \langle v | v_1 \dots v_n \cdot \tilde{\mu}x. \langle v'_1 | (\tilde{\mu}z.c') @ (v'_2 \dots v'_{n'} \cdot \tilde{\mu}y.c) \rangle \rangle \simeq' c_2$, where the last equivalence holds by applying the axiom $\simeq_{\tilde{\mu}\tilde{\mu}}$ with the environment evaluation contexts $D = \langle v | v_1 \dots v_n \cdot \langle \cdot \rangle \rangle$ and $D' = \langle v'_1 | (\tilde{\mu}z.c') @ v'_2 \dots v'_{n'} \cdot \langle \cdot \rangle \rangle$.

(c) *Internal $\tilde{\mu}$ -step*, i.e. $n = 0$ and $c_1 \rightarrow_{\tilde{\mu}} \langle v' | v'_1 \dots v'_{n'} \cdot \tilde{\mu}y.c \{x \leftarrow v\} \rangle = c_2$:

then, $c_0 \rightarrow_{\tilde{\mu}} c_2$ since $x \notin \text{fv}(v') \cup \bigcup_{i'=1}^{n'} \text{fv}(v'_{i'})$.

(d) *External $\tilde{\mu}$ -step*, i.e. $n' = 0$ and $c_1 \rightarrow_{\tilde{\mu}} \langle v | v_1 \dots v_n \cdot \tilde{\mu}x.c \{y \leftarrow v'\} \rangle = c_2$ (recall that $y \notin \text{fv}(v) \cup \bigcup_{i=1}^n \text{fv}(v_i)$): then, $c_0 \rightarrow_{\tilde{\mu}} c_2$.

2. Immediate consequence of Prop. 53.1.

3. Immediate consequence of Prop. 53.1.

4. Immediate consequence of Prop. 53.1. □